E.T.S.I.S. TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA
PLAN 2000

TEMA: Simulación y procesado en comunicaciones.

TÍTULO: Simulación de seguimiento de blancos en sistemas Radar. Parte 2.

AUTOR: Carlos Montalvo Morón

TUTOR: José Enrique Gonzalez Garcia **Vº Bº.**

DEPARTAMENTO: DIAC

Miembros del Tribunal Calificador:

PRESIDENTE: Ruben de Diego Martinez

VOCAL: José Enrique Gonzalez Garcia

VOCAL SECRETARIO: Jose Luis Jimenez Martín

DIRECTOR:

Fecha de lectura: 30/09/2014

Calificación: **El Secretario,**

RESUMEN DEL PROYECTO:

El proyecto consiste en el diseño y la construcción de un prototipo de entorno software para gestionar una simulación de un sistema de radar.

La construcción de tal entorno de simulación requiere potencialmente un esfuerzo muy superior al posible en un PFC por lo que el esfuerzo en este PFC se ha centrado en construir los cimientos básicos del software y preparar las bases de futuros desarrollos.

Se ha utilizado como modelo de las prestaciones ofrecidas por el software con respecto a los usuarios el software de la compañia SAP ya que provee unas prestaciones similares a los usuarios de su software para empresas a las que podría potencialmente este software de gestión de simulaciones de radar a sus propios usuarios.

El esfuerzo de desarrollo se ha centrado en proveer los componentes básicos a partir de los cuales se pueden construir con un mínimo esfuerzo las transacciones que los usuarios ejecutarán para interactuar con el entorno de simulación.

El proyecto consiste en el diseño y estudio de un software cuyas prestaciones estén orientadas a gestionar una simulación de un sistema de radar.

El prototipo de este entorno de simulación se ha realizado en el lenguaje Matlab debido a que inicialmente se considera el más adecuado para el tratamiento de las señales que los sistemas de radar manejan para realizar sus cálculos.

Se ha escogido como modelo el software desarrollado por la compañía SAP para gestionar los E.R.P.s de grandes empresas. El motivo es que es un software cuyo diseño y funcionalidad es especialmente adecuado para la gestión ordenada de una cantidad grande de datos diversos de forma integrada.

Diseñar e implementar el propio entorno es una tarea de enorme complejidad y que requerirá el esfuerzo de una cantidad importante de personas; por lo que este proyecto se ha limitado, a un prototipo básico con una serie de características mínimas; así como a indicar y dejar preparado el camino por el que deberán transcurrir las futuras agregaciones de funcionalidad o mejoras.

Funcionalmente, esto es, independientemente de la implementación específica con la que se construya el entorno de simulación, se ha considerado dividir las características y prestaciones ofrecidas por el sistema en bloques. Estos bloques agruparán los componentes relacionados con un aspecto específico de la simulación, por ejemplo, el bloque 1, es el asignado a todo lo relacionado con el blanco a detectar.

El usuario del entorno de simulación interactuará con el sistema ejecutando lo que se llaman transacciones, que son agrupaciones lógicas de datos a introducir/consultar en el sistema relacionados y que se pueden ejecutar de forma independiente. Un ejemplo de transacción es la que permite mantener una trayectoria de un blanco junto con sus parámetros, pero también puede ser una transacción la aplicación que permite por ejemplo, gestionar los usuarios con acceso al entorno. Es decir, las transacciones son el componente mínimo a partir del cual el usuario puede interactuar con el sistema.

La interfaz gráfica que se le ofrecerá al usuario, está basada en modos, que se pueden considerar “ventanas” independientes entre sí dentro de las cuáles el usuario ejecuta sus transacciones. El usuario podrá trabajar con cuantos modos en paralelo desee y cambiar según desee entre ellos.

La programación del software se ha realizado utilizando la metodología de orientación a objetos y se ha intentado maximizar la reutilización del código así como la configurabilidad de su funcionalidad. Una característica importante que se ha incorporado para garantizar la integridad de los datos es un diccionario sintáctico. Para permitir la persistencia de los datos entre sesiones del usuario se ha implementado una base de datos virtual (que se prevé se reemplace por una real), que permite manejar, tablas, campos clave, etc. con el fin de guardar todos los datos del entorno, tanto los de configuración que solo serían responsabilidad de los administradores/desarrolladores como los datos maestros y transaccionales que serían gestionados por los usuarios finales del entorno de simulación.

This end-of-degree project comprises the design, study and implementation of a software based application able to simulate the various aspects and performance of a radar system.

A blueprint for this application has been constructed upon the Matlab programming language. This is due to the fact that initially it was thought to be the one most suitable to the complex signals radar systems usually process; but it has proven to be less than adequate for all the other core processes the simulation environment must provide users with.

The software's design has been based on another existing software which is the one developed by the SAP company for managing enterprises, a software categorized (and considered the paradigm of) as E.R.P. software (E.R.P. stands for Enterprise Resource Planning). This software has been selected as a model because is very well suited (its basic features) for working in an orderly fashion with a pretty good quantity of data of very diverse characteristics, and for doing it in a way which protects the integrity of the data.

To design and construct the simulation environment with all its potential features is a pretty hard task and requires a great amount of effort and work to be dedicated to its accomplishment. Due to this, the scope of this end-of-degree project has been focused to design and construct a very basic prototype with minimal features, but which way future developments and upgrades to the systems features should go has also been pointed.

In a purely functional approach, i.e. disregarding completely the specific implementation which accomplishes the simulation features, the different parts or aspects of the simulation system have been divided and classified into blocks. The blocks will gather together and comprise the various components related with a specific aspect of the simulation landscape, for example, block number one will be the one dealing with all the features related to the radars system target.

The user interaction with the system will be based on the execution of so called transactions, which essentially consist on bunches of information which logically belong together and can thus be managed together. A good example, could be a transaction which permits to maintain a series of specifications for target's paths; but it could also be something completely unrelated with the radar system itself as for example, the management of the users who can access the system. Transactions will be thus the minimum unit of interaction of users with the system.

The graphic interface provided to the user will be mode based, which can be considered something akin to a set of independent windows which are able on their own to sustain the execution of an independent transaction. The user ideally should be able to work with as many modes simultaneously as he wants to, switching his focus between them at will.

The approach to the software construction has been based on the object based paradigm. An effort has been made to maximize the code's reutilization and also in maximizing its customizing, i.e., same sets of code able to perform different tasks based on configuration data. An important feature incorporated to the software has been a data dictionary (a syntactic one) which helps guarantee data integrity. Another important feature that allow to maintain data persistency between user sessions, is a virtual relational data base (which should in future times become a real data base) which allows to store data in tables. The data store in this tables comprises both the system's configuration data (which administrators and

developers will maintain) and also master and transactional data whose maintenance will be the end users task.

ÍNDICE DEL PROYECTO.

CAPÍTULO 1. INTRODUCCIÓN AL PFC.

Un sistema de radar es un entorno complejo tanto de hardware como de software que permite tanto la detección como el seguimiento de blancos utilizando señales electromagnéticas.

Para diseñar un sistema de radar es necesario poder realizar simulaciones previas de sus distintas características, siendo la cantidad de parámetros que se pueden considerar tan elevada como precisa se desee la simulación.

En este proyecto de fin de carrera, se pretende diseñar un sistema de software que permita la gestión de parámetros y resultados de simulaciones de radar. Inspirándose en el modelo creado por el software de gestión empresarial SAP[i], se pretende habilitar un entorno gráfico de tipo transaccional que permita a distintos usuarios manipular las configuraciones, parámetros y resultados de distintas simulaciones.

Para implementar este esbozo inicial de entorno de simulación se ha utilizado el lenguaje de programación Matlab. Esto se debe, a que podría ser el más indicado para manejar los cálculos numéricos que se deben realizar en las distintas simulaciones. Sin embargo, debido a lo limitado del lenguaje en otros aspectos, podría ser necesario utilizar otro lenguaje más apropiado como núcleo del software, haciendo llamadas específicas a una "capa" de Matlab solo cuando sea realmente apropiado.

Análogamente, para almacenar tanto las configuraciones y resultados de las simulaciones, como los propios datos necesarios para el entorno de simulación en sí, se ha abstraído una capa de base de datos. En este esbozo inicial se utilizan simples ficheros de Matlab. Sería necesario para mejorar el rendimiento y la gestión, así como para permitir la utilización recursiva de la herramienta por varios usuarios sustituirlo cuando fuese posible por un servidor de base de datos o algún otro recurso equivalente de gestión de datos.

Con respecto a la funcionalidad que presenta el entorno de simulación, se ofrece una pequeña muestra inicial de todo lo que potencialmente se puede ofrecer, esperando que a partir de la base aquí ofrecida, en posteriores proyectos se puedan ir añadiendo parte a parte distintos componentes o mejorando los ya existentes, todo ello aprovechando la "modularidad" de la que se ha querido dotar el software. Para este efecto, se ha utilizado una nomenclatura para los distintos objetos de desarrollo que los separa en bloques, siendo distintivo de cada uno de ellos que sus nombres comiencen con un prefijo de 3 letras, siendo las dos primeras BL y la tercera una letra o un digito que indique el componente relacionado. Por ejemplo, BL1 será el prefijo para las funciones relacionadas con el bloque 1 que es el asociado a todos los posibles parámetros del blanco. Si el alcance de lo desarrollado agotase esta nomenclatura sería inmediato habilitar prefijos de 4 o más caracteres para poder continuar ampliando el entorno.

Básicamente se ha pretendido proveer de un punto de partida sobre el cual construir un entorno de simulación de radar de altas prestaciones (al estilo de lo que SAP ha conseguido para el software empresarial). El que solo se presente una primera aproximación u esbozo se debe a la imposibilidad de abarcar un esfuerzo de considerable magnitud (SAP es la segunda empresa en tamaño de software del mundo) por una sola persona. Sin embargo, está en el ánimo del diseño del software el que añadiendo el trabajo posterior de profesores/alumnos se pueda disponer de una herramienta realmente útil para la enseñanza y diseño de los sistemas de radar.

CAPÍTULO 2. INTRODUCCIÓN TEÓRICA.

En este capítulo se presenta una introducción teórica al concepto de sistema radar y se describen y clasifican los distintos componentes y aspectos que en un principio se consideran susceptibles de ser incorporados a una herramienta de simulación.

2.1. Introducción al concepto de radar.

2.1.1. Definición.

El término RADAR proviene de las iniciales de "**RA**dio**D**etection**A**nd **R**anging". Un sistema de RADAR permite buscar objetos en un determinado espacio mediante la emisión y recepción de energía electromagnética radiada. El funcionamiento se basa en emitir y recibir energía en el espacio, y estimar características de "blancos" a partir de la energía electromagnética que se supone que estos reflejan.

2.1.2. Clasificación.

Los sistemas radar existentes son muy diversos y para clasificarlos se utilizan criterios bastante variados:

- Por su localización.
- Por su misión.
- Por sus características técnicas.
- Etc., etc.

Una distinción que se hace entre ellos es por el tipo de onda electromagnética que se utiliza, lo que se denomina "forma de onda". Se distingue si la emisión de energía es continua o si por el contrario, se emiten ráfagas de pulsos de energía.

También se suelen clasificar, en función de otro parámetro clave de la onda electromagnética utilizada, su frecuencia principal. En función del valor de esta frecuencia (llamada frecuencia de la portadora), los sistemas de radar se asocian a una serie de bandas o intervalos.

2.2. Consideraciones generales sobre el alcance "razonable" de una simulación.

2.2.1. Motivo de la restricción de alcance de una simulación.

Para poder diseñar una herramienta de simulación de radar, es necesario asumir una serie de características como punto de partida. No es posible empezar a construir la herramienta si no se han asumido donde irán los "pilares". La modificación de estas consideraciones básicas conllevaría una modificación sustancial del enfoque de la herramienta pero siempre será posible aunque costa de un gran esfuerzo.

2.2.2. Clasificación del sistema radar a simular.

Una forma de restringir el alcance de la simulación es limitarlo a ciertas "clases" de sistemas radar. Asumir una u otra clasificación implica que cualquier cambio en el alcance de la simulación que añada una nueva clase o característica supondrá un esfuerzo importante en desarrollo, pero es algo inevitable a priori si se quiere tener un punto de referencia desde el que comenzar el desarrollo. En un principio se ha considerado razonable asumir las siguientes restricciones respecto:

- Con respecto a la localización, se considerará que el sistema de radar estará fijo en tierra, esto simplificará considerablemente los cálculos y los parámetros ya que lo único que se puede considerar como móvil son los propios blancos. La alternativa a esto será considerar, tanto el radar como los blancos, móviles para lo que habría que introducir una mayor complejidad.
- Otra decisión, también asociable a la localización es considerar un sistema radar monoestático, es decir, que tanto el emisor como el receptor se encuentran en el mismo punto. Esto deja como ampliación de la funcionalidad del entorno de simulación, la simulación de radares biestáticos, aquellos en los que el emisor y el receptor se encuentran en puntos distintos.
- Con respecto a la forma de la señal, se considerará un radar pulsado, es decir que emite a ráfagas. La ampliación a esto sería considerar emisión de ondas continuas, en las que la detección se llevaría a cabo contabilizando variaciones en frecuencia.
- También relativo a la señal se debe de elegir la relación entre longitud de onda y tamaño/distancia de los blancos detectados. Según la magnitud de esta relación se considera que se está operando en distintas "zonas". Las implicaciones de trabajar en las distintas zonas es muy importante en lo relativo a las leyes de la física y las simplificaciones que cabe aplicarles. Trabajar en simulaciones bajo la hipótesis de que el blanco o la distancia no es mucho menor que la longitud de onda no tiene mucho sentido práctico, por lo que se restringirán los parámetros de la simulación para asegurarse que se trabaje en la zona óptica, que es aquella en que el blanco es lo suficientemente pequeño y está a la suficiente distancia de la antena como para que se consideré que las ondas funcionan con las leyes de la óptica (i.e., principalmente la reflexión).

2.2.3. Otras restricciones al alcance de la simulación.

Las siguientes consideraciones también son útiles, inicialmente, como guía para restringir el alcance del entorno de simulación:

- Otra decisión relevante es el tipo de detección. Básicamente, existen dos formas de estimar blancos. Los sistemas radar, normalmente, utilizan estas dos formas en mayor o menor grado: una, es la detección basada en rango; y otra, la basada en frecuencia o Doppler (llamada así por que se utiliza la información creada por el efecto del mismo nombre). Como punto de partida se elige la detección basada en rango; porque si bien es de uso más limitado, presenta menor complejidad. Ampliar las posibilidades de simulación a detección en frecuencia sería bastante prioritario, dado lo extendido de su uso.
- Otro aspecto, también bastante relevante (para el compromiso realismo vs complejidad del entorno de simulación), es el proceso de modulación/demodulación de la señal que se produce tanto al emitirla como recibirla. Este proceso está motivado porque realmente la información que es necesario procesar se encuentra en una estrecha banda de frecuencias, pero para mejorar la transmisión es necesario modular esta pequeña banda a frecuencias elevadas. Para una primera aproximación a la simulación se considera razonable ignorar este proceso y suponer que se está trabajando en banda base. No obstante, posteriormente, se podrán añadir parámetros que caractericen este proceso.
- Existen toda una serie de procesos a nivel de circuitería que afectan a la calidad de las señales electromagnéticas tanto antes de su emisión como tras su recepción. En esta primera instancia se obviarán estos procesos internos en la circuitería de la antena, como también se obviará cualquier limitación física que pudiese darse a la velocidad con la que antena es capaz de variar su apuntamiento. Queda a futuro considerar añadir estos detalles.
- La calidad de la señal también puede ser afectada mientras está viajando por el aire. Existen multitud de factores que afectan sobre la señal de muy diversas formas. Claramente, solo un puñado de estos factores podrán considerarse en una primera aproximación, pudiéndose añadir posteriormente al entorno de simulación diversos factores según se vayan considerando y estudiando.
- Otros ámbitos a considerar surgen cuando se consideran las numerosas técnicas de procesado de la señal que permiten mejorar la calidad y fiabilidad del proceso de detección. Incorporar las más sencillas inicialmente, y en subsiguientes mejoras ir añadiendo procesos que impliquen una cantidad mayor de parámetros, es la mejor forma de dar una forma inicial al entorno de simulación.

2.3. Asignación de los distintos aspectos de la simulación a bloques.

Para poder organizar toda la funcionalidad que se va desarrollando en el entorno de simulación es útil crear una clasificación por bloques que agrupen las funcionalidades y características asociadas a un aspecto de la simulación. Esta clasificación es arbitraria e incompleta, pero sirve como punto de partida.

A modo de esbozo, los siguientes bloques, pueden valer para agrupar los distintos componentes o aspectos del entorno de simulación:

- Bloque 0: Parámetros globales de la simulación.
- Bloque 1: Blanco.
- Bloque 2: Transmisor y del receptor (antenas).
- Bloque 3: Señal.
- Bloque 4: Entorno de propagación.
- Bloque 5: Proceso de detección.
- Bloque 6: Seguimiento de blancos.
- Bloque 7: Predicción del movimiento de blancos.
- Bloque 8: Seguimiento simultáneo de múltiples blancos.

El entorno de simulación debería modularizarse de tal forma que permita añadir nuevos bloques o ampliar la funcionalidad comprendida en cada una sin que sea necesaria una redefinición de los bloques ya existentes.

A continuación se hace un breve recorrido de los detalles que se pueden considerar para cada uno de los bloques:

Bloque 0.

A la hora de realizar simulaciones de sistemas radar, se encuentra el problema de que los resultados de la simulación no pueden ser calculados ni mostrados como una función continua del tiempo, sino que dado que se utilizan computadores como herramientas, es necesario trabajar con señales discretas que emulen o sustituyan a las señales continuas que existen en la realidad.

La duración temporal de la simulación, el número de muestras y la frecuencia de muestreo con que estas se toman son pues parámetros globales que afectan a todos los aspectos de la simulación.

A la hora de elegir estos parámetros en las simulaciones siempre hay que considerar que se está en un compromiso:

Cuanto mayor sea la duración de la simulación; más probabilidades habrá de observar el proceso de detección y seguimiento de los distintos blancos con precisión. Igualmente, cuanto mayor sea el número de muestras que se tomen; mayor será la exactitud y veracidad de la simulación; principalmente, porque muchas de las variables que se utilizan son variables estadísticas, y el comportamiento de estas, se aproxima más a la realidad cuanto mayor es el número de muestras. Por contra, el procesado que exige la simulación es directamente proporcional al número de muestras; por lo que un número excesivo de éstas, conllevaría un tiempo de computación y una utilización de memoria elevados; y si la magnitud de muestras es demasiado elevada, podría sobrepasarse la capacidad del procesador.

Otro parámetro que se puede considerar global a la simulación es el máximo rango de detección. El máximo rango de detección viene delimitado por el concepto de máximo rango de detección sin ambigüedades.

Cuando se emiten pulsos periódicamente y se espera recibir los reflejos de estos pulsos, cabe la posibilidad de que el blanco esté tan lejos que la respuesta a un primer pulso llegue después de haber emitido un segundo. En este caso, es imposible decir, si el pulso recibido corresponde al reflejo del primer pulso (y el blanco estaría muy lejos) o por el contrario corresponde al reflejo del segundo pulso recién emitido (y el blanco estaría mucho más cerca).

Por lo tanto el rango de detección en el cuál se puede considerar que no se está sujeto a ambigüedad es proporcional a la distancia temporal con que se emiten los pulsos, esto es el PRF (Pulse Repetition Frecuency en inglés). Este parámetro se puede considerar global a la simulación.

Para contrarrestar el efecto de la ambigüedad se han desarrollado técnicas consistentes en la utilización alternativa de distintas PRFs al emitir. Esto permite que el rango real de detección sin ambigüedades aumente en múltiplos de las frecuencias utilizadas. Añadir estas técnicas a la capacidad del entorno de simulación es una de las posibles mejoras de sus prestaciones.

Por otro lado en paralelo al concepto de rango, existe el concepto de resolución del sistema radar tanto en rango como en ángulo. La resolución es la capacidad de diferenciar rangos cercanos entre si y no confundirlos con uno solo. La resolución es un buen candidato a parámetro global de la simulación.

La resolución en rango depende de cual sea la duración del pulso, lo que se conoce como ciclo de trabajo. Si se emiten pulsos con una duración corta con respecto al intervalo entre pulso y pulso, aumenta la posibilidad de que existiendo dos blancos juntos, cuando el pulso alcance el segundo blanco tras haber pasado por el primero, ya haya comenzado la reflexión en el primero, si por el contrario el pulso es más largo, no será posible diferenciar la reflexión correspondiente al primer blanco con la correspondiente al segundo, siendo todo un continuo. Existen técnicas de tratamiento de la señal, que jugando con la frecuencia permiten emitir pulsos con el efecto de aparentar ser más cortos de lo que de otra forma sería posible. Estas técnicas de compresión de pulsos también son susceptibles de ser incorporadas a la simulación.

La resolución en ángulo depende sin embargo, no de la propia señal, sino de las características de las antenas emisoras y receptoras. Las antenas emiten/reciben energía mejor en ciertos intervalos de ángulos llamados lóbulos, es el tamaño de estos lóbulos lo que limita la resolución en ángulo de un sistema radar. Una vez fijada una resolución en ángulo está debería limitar el tipo de antenas que es posible utilizar.

Bloque 1.

La simulación de radar debería estar diseñada de tal forma que permita caracterizar uno u varios blancos según se desee. Es decir, debería ser posible definir una serie de características para un blanco y asociarlas una identificación o código. Luego posteriormente en la simulación se indicarían uno o varios de estos códigos para identificar las características de los blancos.

Hay dos características principales que un blanco presenta en un sistema radar (es posible, sin duda, encontrar más aspectos del blanco y estos se podrían también incluir en la simulación): su posición respecto a la antena transmisora/receptora y su sección radar.

Con respecto a la posición del blanco, esta tendrá un valor en cada instante de tiempo de la simulación y estos valores vendrán determinados por los siguientes parámetros:

- Posición inicial e instante inicial: la posición inicial será donde el blanco comience su trayectoria, mientras que el instante inicial será el momento o el número de muestra temporal a partir de la cual el blanco comienza su trayectoria.
- Velocidad inicial: velocidad con la que el blanco inicialmente comienza su trayectoria (a partir del instante inicial).
- Aceleración: esta magnitud vectorial debería ser una función del tiempo es decir tener un valor determinado para cada muestra temporal. Para determinarla, se pueden seguir tres aproximaciones, indicarla muestra a muestra, calcularla mediante una fórmula, o calcularla a partir de una función aleatoria. También sería útil que se pudiese determinar con una combinación de estos métodos.

Como herramienta útil para organizar la simulación sería bueno agrupar estos datos en “trayectorias” y crear un “maestro de trayectorias” que luego pudiese ser asociado a los blancos. La utilidad de gestión de este maestro de trayectorias podría incluir entre otras capacidades, aparte de la propia gestión de los parámetros, la representación de la propia trayectoria en gráficas espaciales.

Con respecto a la sección radar que presente el blanco, que viene a representar como de bien refleja el blanco las ondas transmitidas por el radar, esta se podía gestionar de dos formas en el entorno de simulación: de forma determinista o estadística.

Tratar la sección radar de forma determinista implicaría la aproximación más sencilla, es decir, habría que elegir la forma del blanco de entre una serie de formas predeterminadas (punto, esfera, elipsoide, cono truncado,..) y luego una serie de magnitudes apropiadas a cada forma (ángulos, proporciones, etc.). Estas magnitudes, podrían ser constantes, indicarse muestra temporal a muestra temporal o determinarse a partir de una fórmula o incluso a partir de una función aleatoria.

La otra posibilidad, tratar la sección radar de forma estadística, implicaría utilizar el modelado de Swerling que se maneja en los sistemas de radar a este efecto. Habría que elegir la función Swerling que se considera se aproxima más al comportamiento del blanco y luego indicar los parámetros que requiera la función densidad de probabilidad asociada al tipo Swerling elegido.

Por cada blanco además habría que indicar un parámetro de “Área del blanco” que aplicada proporcionalmente a los parámetros escogidos según el tipo de sección radar determinaría la sección radar presentada por éste. Hay que considerar que esta magnitud

debería quedar limitada por la longitud de onda de la señal utilizada, en el sentido que ya se explicó anteriormente: la proporción entre esta área y la longitud de onda, debe ser lo suficientemente grande como para que se pueda considerar que se está funcionando en la zona óptica.

La polarización de las ondas transmitidas influye en la sección radar que presenta un blanco, pero debido a su complejidad, se estima que es un efecto cuya inclusión en el entorno de simulación es mejor retrasar para futuras ampliaciones.

Bloque 2.

Una de las partes fundamentales de los sistemas de radar son los transmisores y receptores de las señales electromagnéticas. A su vez los componentes más importantes tanto en los transmisores como en los receptores son las antenas: ellas son las que se encargan de transformar la energía guiada en energía radiada y viceversa.

Por lo tanto gran parte de la parametrización asociada a este blanco estará determinada por los características de las antenas, aunque previamente (o posteriormente) a la antena propiamente dicha existen una serie de componentes que también son necesarios para la transmisión (o recepción) y que también deberían tener su parametrización asociada en el entorno de simulación.

Es posible utilizar la misma antena tanto para transmitir como para recibir, pero también es posible utilizar distintas antenas. Como simplificación inicial se puede considerar que la antena es la misma para las dos funciones, pero el entorno de simulación debería fácilmente permitir incorporar la posibilidad de que la antena para transmisión no sea la misma que la antena para recepción.

Como el alcance de la simulación de radar se prevé que incluya no solo la detección de un blanco sino también su seguimiento, desde el principio se deben prever dos conjuntos de pares transmisor/receptor. Esto es así porque las características más adecuadas para realizar la detección de un blanco no son las más adecuadas para realizar su seguimiento. Permitiendo ya desde el principio configurar dos conjuntos diferenciados, habilitamos la simulación de ambas modalidades de funcionamiento del radar, y en caso de que se quisiese simular un sistema en que no hay diferencia en los componentes que realizan ambas funciones, no habría más que configurar los dos componentes con los mismos parámetros.

Las propias antenas son elementos muy complejos que exigen multitud de parámetros para poder acomodar simulaciones realistas. La forma en que esta complejidad se puede afrontar en un entorno de simulación es utilizando técnicas de composición, i.e., obtener funcionalidad compleja a partir de la agregación de bloque de funcionalidad más sencilla. Para este propósito se pueden dividir las antenas en tres tipos: elementales, sencillas y compuestas.

Las antenas elementales serán antenas predefinidas en el entorno de simulación, que quedarán definidas por un diagrama de radiación función de unos pocos parámetros sencillos, se las puede considerar como moldes. Las antenas sencillas serán una muestra específica con unos parámetros específicos, que vayan asociados a una o varias antenas elementales. Por último, las antenas compuestas corresponderán a disposiciones en el espacio de conjuntos de antenas sencillas con una geometría específica. Las antenas compuestas equivalen y pretenden simular lo que se conoce como arrays de antenas.

Las antenas son un componente de la simulación muy adecuado para ser gestionado como “datos maestros”. A partir de las antenas elementales, se podrían registrar y gestionar en el sistema el número que fuese necesario de antenas sencillas y compuestas con sus respectivos códigos de identificación. A la hora de caracterizar una simulación, solo habría que elegir entre las antenas previamente creadas como datos maestros, facilitando así la utilización de las mismas antenas en distintas simulaciones. Las transacciones de gestión de las antenas podrían llevar asociadas además de los valores de los parámetros asociados, diagramas tanto de la antena física como de su diagrama de radiación.

Otro concepto asociado a las antenas que se puede incorporar a la simulación es su capacidad de variar su apuntamiento, ya sea porque se consideré que este se modifica mecánicamente o porque se consideré un apuntamiento electrónico. Este aspecto de las antenas, estaría directamente relacionado con la función que realizan para las tareas de seguimiento, que requieren que la antena esté constantemente apuntando al blanco. Posiciones límite, así como velocidades angulares máximas o aceleraciones serían parámetros a configurar.

Tanto el transmisor como el receptor no consisten solo de una antena, sino que tienen asociados una circuitería interna que permiten su funcionamiento. Sería posible añadir una cantidad elevada de parámetros para caracterizar su rendimiento, pero para un comienzo, puede bastar con un solo parámetro, el de eficiencia, que sería simplemente un coeficiente entre 0 y 1 midiendo como de bien se transmite la energía de la fuente a la antena o viceversa, de la antena al receptor. También se pueden considerar los amplificadores de manera sencilla, mediante solo dos parámetros, su temperatura equivalente de ruido y su ganancia.

Bloque 3.

Las señales que se emiten definen en gran medida las prestaciones de los sistemas de radar. Una parte de los parámetros de configuración asociados a este bloque deberían venir restringidos por lo fijado por los parámetros globales de la simulación asociados al bloque 0. Se puede considerar que hay dos tipos de datos relacionados con la señal emitida, los asociados a la parte de la señal que lleva la información (señal en banda base) y los asociados a la parte de la señal que ayuda a transportar la información (señal portadora).

Para transmitir la banda de frecuencias de donde realmente se extrae la información útil para la detección, muchas veces es necesario transportar esta “banda base” a una frecuencia muy superior llamada frecuencia de la onda portadora o frecuencia portadora. Esta frecuencia portadora, debería ser uno de los parámetros más importantes de la simulación.

La frecuencia de la portadora y su correspondiente longitud de onda, gobiernan muchos de los aspectos de la transmisión y recepción de las ondas. Por lo tanto, el valor de este parámetro debería restringir el valor de otros muchos parámetros. Por ejemplo, las dimensiones de las antenas y los blancos, o el efecto de los fenómenos atmosféricos deberían ser función de la longitud de onda de la portadora.

La frecuencia de la portadora también automáticamente clasifica el sistema radar según la banda en que se encuadre. Existen varios esquemas de clasificación con sus nombres y rangos, este es el del IEEE:

Band	Frequency range
HF Band	3 to 30 MHz
VHF Band	30 to 300 MHz
UHF Band	300 to 1000 MHz
L Band	1 to 2 GHz
S Band	2 to 4 GHz
C Band	4 to 8 GHz
X Band	8 to 12 GHz
Ku Band	12 to 18 GHz
K Band	18 to 27 GHz
Ka Band	27 to 40 GHz
V Band	40 to 75 GHz
W Band	75 to 110 GHz
mm Band	110 to 300 GHz

Con respecto a la señal en banda base, su caracterización, para las limitaciones fijadas de trabajar solo con señales pulsadas, estará restringida por los valores fijados para los parámetros globales de la simulación.

Por ejemplo, el máximo rango de detección sin ambigüedades fijará el PRF o periodo de repetición entre pulsos. Se podría considerar en este bloque implementar la funcionalidad necesaria para simular la utilización de múltiples PRFs.

También la resolución en rango del sistema de radar, va a limitar el ciclo de trabajo de la señal. Cabe relacionado con esto, incluir en este bloque parámetros para simular la utilización de técnicas de compresión de la señal, que permiten disminuir a efectos de procesado de la señal el ciclo de trabajo de esta.

Otra de las características que debería quedar definida en este bloque es la potencia de la señal transmitida, según sale de la fuente, hay que tener en cuenta que la magnitud de esta potencia debería estar sujeta a un intervalo realista para sistemas de radar reales, que debido a su tamaño están sujetos a limitaciones en ella.

Bloque 4.

Desde que se transmite la señal hasta que se recibe su reflejo, ésta queda sujeto a una serie de fenómenos que obstaculizan el funcionamiento de un sistema de radar. Este bloque agrupa todos estos fenómenos que de un modo u otro influyen en la calidad de la señal recibida.

Uno de los principales factores que impactan la calidad de la señal recibida, es que aparte de la propia señal se reciben otras reflexiones o retornos no deseados provenientes de otras fuentes. Estos retornos no deseados normalmente se clasifican en dos categorías, que responden a los términos “Clutter” y “Jamming”.

Bajo el término “clutter” se engloba los retornos causados por retornos de objetivos aleatorios que no resultan de interés para el sistema de radar. Árboles, olas, pájaros, todos ellos producen reflexiones que interfieren con las reflexiones provenientes del objetivo en el que el radar fija su interés. Normalmente todas estas fuentes de clutter comparten la característica de que su velocidad relativa al radar es nula o muy pequeña, por lo tanto se utilizan técnicas de filtrado en frecuencia que directamente ignoran los retornos de blancos cuasi estacionarios. Como estas técnicas de filtrado se basan en la detección en frecuencia, y ésta en un principio no se incorpora al entorno de simulación, incorporar estos fenómenos no se puede hacer más allá de considerar unos coeficientes sencillos como relación entre potencia de señal y potencia del clutter. Una vez que se incorporase a la simulación, la funcionalidad de detección por frecuencia o Doppler, sí que sería posible incorporar parámetros para simular el proceso de filtrado en frecuencia de las señales provenientes del clutter.

Contrariamente al clutter que consiste en la recepción de señales aleatorias no deseadas, las señales recibidas consideradas como jamming, son emitidas intencionadamente con el objetivo de degradar el funcionamiento del sistema radar. Combatir el jamming requiere un procesado muy complejo de la señal (emisión en múltiples bandas de forma aleatoria por ejemplo) y debería ser uno de los aspectos que se incorporasen al entorno de simulación solo cuando esté ofreciese la caracterización de otros muchos aspectos más sencillos y esenciales. Si se desea se puede tratar este fenómeno inicialmente, a la manera del clutter con un sencillo coeficiente que indicase la relación entre la potencia de la señal de interés y la potencia del jamming.

Existen diversos atmosféricos que provocan la atenuación de la señal emitida, la magnitud en que estos fenómenos afectan a la señal es función de la distancia que recorre la señal, la zona de la atmósfera que atraviesa, la propia frecuencia de la portadora, la polarización, etc. Normalmente la incidencia de estos fenómenos está tabulada o aproximada mediante fórmulas. Se pueden integrar estos fenómenos al entorno de simulación, simplemente caracterizando cada uno de ellos con los parámetros de los cuales dependen.

Otros fenómenos que afectan la propagación de la señal, y que son susceptibles de ser incorporados al entorno de simulación son:

- Refracción: debido a la variación gradual de la densidad de la atmosfera se produce un fenómeno de refracción sobre la onda de transmitida que tiene el efecto de curvar la trayectoria de ésta curvándola y alargándola. Además al recibir la señal se tiene que considerar este fenómenos para ajustar la posición del blanco, ya que la onda no viajaría en línea recta del blanco al radar sino siguiendo una trayectoria curvada.

- Reflexión: las ondas emitidas no solo llegan tras reflejarse directamente en el blanco, sino que previamente pueden haberse reflejado en el suelo. Según la diferencia de trayectorias entre la onda viajando de forma directa y viajando con reflexiones los distintos retornos podrían añadirse, ayudando a la detección o cancelarse, dificultándola.
- Difracción: este otro fenómeno óptico también influye en la trayectoria de estas haciendo que rodeen obstáculos para llegar al detector y modificando la trayectoria ideal. La existencia de obstáculos entre el blanco y el detector, alteraran las posibilidades de detección de este.

Bloque 5.

Una vez que se ha recibido la señal y está ha pasado por un primer amplificador que ha aumentado su nivel pero también la ha añadido ruido térmico se efectúa el proceso de detección propiamente dicho.

Para facilitar su detección la señal se pasa siempre por lo que se llama filtro adaptado, que consiste en uno ajustado completamente a la señal emitido. Este filtro, dejaría fuera toda aquella parte de la señal recibida que no correspondiese a las frecuencias de la señal emitida, eliminando en el proceso un montón de energía de ruido no deseada. Se pueden añadir una serie de parámetros para caracterizar esta fase, pero inicialmente, como simplificación es mejor considerar un filtro perfectamente adaptado.

Para caracterizar el proceso de detección uno de los parámetros que se debe poder elegir es la probabilidad de falsa alarma, que mide cuál es la probabilidad de que a partir de un nivel de señal recibida se considere que existe blanco, cuando realmente la energía de señal recibida no corresponde a la reflexión de señal emitida sino a otras fuentes no deseadas.

A partir de la probabilidad de falsa alarma, de la potencia de la señal emitida y del rango de detección del sistema se determinará un umbral de detección. Cuanto mayor sea la potencia de señal emitida mayor debe ser ese umbral; por el contrario, también hay que considerar que cuanto más lejano sea el rango en el que se desean detectar blancos, la señal recibida de la reflexión será menor y menos debe ser el umbral. Por último, cuanto menor sea el umbral de detección mayor será la probabilidad de falsa alarma, ya que, existirán más posibilidades de que solo la energía de señales indeseadas sea suficiente para rebasarlo.

Una mejora que se suele incluir en el procesado de señal de los sistemas de radar es ajustar el umbral de detección a la potencia recibida; esta técnica se conoce en inglés como CFAR (Constant False Alarm Rate). Básicamente lo que se hace es ir aumentar el nivel del umbral de detección si la potencia recibida está siendo alta, o disminuirlo si está siendo baja. Por ejemplo, si el blanco es muy lejano y llega muy poca potencia, con esta técnica automáticamente se disminuiría el umbral, posibilitando su detección. Al contrario, si el blanco esté muy cercano, y llegase mucha potencia, se aumentaría el umbral, dificultando la posibilidad de que una falsa alarma lo superase. Utilizando esta técnica, se intenta que la probabilidad de falsa alarma no sea función de la distancia del blanco. Se puede considerar añadir esta técnica al entorno de simulación, permitiendo contrastar la diferencia en los resultados que se da cuando se utiliza y cuando no se utiliza.

Otra técnica susceptible de ser incorporada al entorno de simulación es la integración de pulsos en la detección. Esta técnica se basa en el siguiente hecho: la energía que se recibe por cada pulso tiene dos componentes, la asociada a la energía emitida y la asociada al ruido o

energía no deseada; mientras que la energía deseada debería ser coherente entre pulso y pulso recibido, no ocurre lo mismo con la energía asociada al ruido. Por lo tanto si en vez de tomar decisiones de detección aplicando un umbral pulso a pulso, se toman cada n pulsos, integrando la energía de cada pulso y ajustando el umbral convenientemente, se mejorará mucho la relación entre señal deseada y no deseada y por tanto el proceso de detección. El entorno de simulación podría para simplificar las cosas simplemente permitir elegir si se utiliza esta técnica o no y cuantos pulsos se integran (la mejora en la detección no aumenta ad-infinitum con el número de pulsos, sino que llega un punto de saturación a partir del cual no se mejora la detección aunque se integren más pulsos).

Bloque 6.

El entorno de simulación, como ya se ha expuesto, podría incluir no solo la simulación de la detección de blancos, sino también la simulación del seguimiento de los blancos detectados. La caracterización de esta faceta se agrupa en este bloque.

El seguimiento de blancos se suele realizar mediante una técnica conocida como TWS (Track While Scan en inglés) que básicamente consiste en estar escaneando e intentando detectar blancos continuamente y una vez que se detecta un blanco dedicarle una atención especial y realizarle un seguimiento específico.

Para simular esto, una aproximación consiste en mantener dos sistemas de antenas independientes, uno que se encargue de la detección y otro del seguimiento.

El sistema encargado de la detección, a su vez estaría compuesto de un par de antenas, una que barriese el espacio en elevación y otro que lo barriese en azimut. Por su parte, el sistema de seguimiento estaría integrado por una batería de antenas (cuyo número y características deberían ser configurables) que se irían asignando al seguimiento de los distintos blancos detectados.

Mientras que las antenas encargadas de la detección deberían barrer simultáneamente amplias porciones del espacio, las encargadas del seguimiento, justamente al contrario deberían ser lo más direccionales posibles.

Estos dos sistemas de antenas pretenden simular o sustituir la operativa real de sistemas radar, que en realidad funcionan con una sola antena capaz de detectar simultáneamente en varias direcciones a la vez que puede ir ajustando el apuntamiento de sus “lóbulos” según lo requieran los blancos. Este tipo de antenas se basan a su vez en arrays de antenas cuya alimentación se varía continuamente según se necesite. Una mejora importante en el realismo del entorno de simulación sería añadir la posibilidad de directamente poder configurar este tipo de antenas.

Otros parámetros que se deberían especificar, serían valores límites, para las áreas de búsqueda del sistema. Por ejemplo, se podrían especificar mediante rangos de ángulos en elevación y azimut las zonas en las que se supone el sistema activo.

Las antenas de seguimiento funcionan apuntándose continuamente hacia el blanco, y, en función de los retornos que reciben del blanco deben ir estimando cual debe ser su posición futura. Existen varias formas de realizar este seguimiento:

- La antena puede tener un lóbulo que alternativamente rote entre cuatro direcciones perpendiculares entre si y formando un ángulo con la línea

imaginaria que une la antena con el blanco (línea de visión, LOS, Line Of Sight en inglés).
- La antena puede tener un lóbulo que rote continuamente con una velocidad angular formando un ángulo con la línea de visión.
- La antena también puede emitir simultáneamente cuatro lóbulos perpendiculares entre sí que formen un ángulo con la línea de visión.

El entorno de simulación podría ofrecer manejar cualquiera de estas técnicas teniendo en cuenta los parámetros que exigirían en cada caso y la complejidad que requeriría su implementación.

Como el proceso de seguimiento implica un cambio en el apuntamiento de las antenas, aquí habría que tener en cuenta las configuraciones mencionadas en el bloque 2 sobre sus limitaciones.

Bloque 7.

Otra de las prestaciones que pueden presentar los sistemas de radar, y que es susceptible de ser incorporada al entorno de simulación es la predicción de la trayectoria de los blancos.

Los sistemas de radar intentan predecir la trayectoria de un blanco siempre a partir de datos sobre la posición del blanco en instantes anteriores al actual. Básicamente, cuando un blanco se detecta se va guardando asociado a él un registro con las posiciones donde se le ha ido localizando. Aplicando una fórmula sobre estas magnitudes se calcula cuál será más probablemente la posición del blanco en el futuro.

Existen diversos métodos de cálculo de esta posición futura, más o menos sofisticados según la precisión que se desee obtener en la predicción. Estos distintos métodos pueden ser incorporados para su selección en el entorno de la simulación.

El cálculo de la posición futura se realiza como si de procesado de señales se tratase, y los distintos métodos de cálculo se implementan mediante filtros, de los que básicamente se utilizan dos tipos: aquellos de ganancia constante y aquellos de ganancia variable.

Los filtros de ganancia constante son aquellos cuyos coeficientes no varían a lo largo del proceso de predicción y se diferencian unos de otros por su relativa complejidad o número de etapas (alpha-beta, alpha-beta-gamma, etc.).

En contraposición a estos, en los filtros de ganancia variable los coeficientes se calculan a su vez en función de los valores de señal de momentos anteriores, de esta forma, la respuesta del filtro se va adaptando al tipo de señal que recibe. Los principales representantes de este tipo de filtros se denominan filtros Kalman.

La funcionalidad principal asociada a este bloque sería la de elegir distintos de filtros y dar valores a sus parámetros, y permitir contrastar la trayectoria real de un blanco con la trayectoria predecible mediante este tipo de técnicas.

Bloque 8.

Los sistemas de radar que realizan labores de seguimiento de blancos suelen ser capaces de gestionar varios blancos simultáneamente. Esta es una característica que se puede agregar al sistema a modo de escalado de las funcionalidades asociadas al bloque 6.

Los sistemas de radar que permiten seguimiento de múltiples blancos, funcionan en esencia llevando un registro por cada blanco detectado donde informan de sus distintas características. Por ejemplo, pueden ir guardando la información relativa a su posición, velocidad, sección radar, etc. Cuando se detecta un blanco, lo que hacen es calcular si este blanco no tiene unas características que coincidan con las de un blanco ya detectado. Si las características no coinciden se considera que es un blanco nuevo y se crea un nuevo registro específico para "seguirle la pista", si por el contrario las características coinciden o se aproximan mucho se considera que se trata del mismo blanco que ya se tenía detectado.

El cálculo a partir del cual se decide sobre si se trata de un blanco no detectado, se hace a base de correlar los datos previos de otros blancos con los actuales. El entorno de simulación puede ofrecer pues configurar el número de blancos que simultáneamente se pueden seguir así como distintas fórmulas de correlación para determinar cuándo se está ante un blanco nuevo y cuando se trata de un blanco ya detectado.

CAPíTULO 3. DISEÑO DEL ENTORNO DE SIMULACIÓN.

En este capítulo se expone como se han diseñado los componentes básicos del entorno de simulación. La motivación básica detrás de la construcción ha sido su máxima configuración y reusabilidad. Esta motivación, ha tenido, como consecuencia negativa en algunas facetas, una eficiencia menor que la que habría sido posible. Queda a futuro pues, corregir esta falta sin renunciar a las ventajas que presentan una alta configurabilidad y modularización.

3.1. El paradigma.

Como ya se ha mencionado en la introducción, se ha pretendido reproducir la estructura y funcionalidad de un entorno SAP para una herramienta de simulación de radar.

El entorno SAP es altamente complejo, y se han requerido decenas de años y cientos de miles de programadores para poderlo construir. Obviamente, construir algo a esa escala no es una meta realista, e incluso construir algo parecido a una escala más modesta es algo que se escapa a la capacidad de trabajo de una persona y requerirá la colaboración y el esfuerzo sucesivo de un conjunto amplio de personas.

Se ha pretendido pues en este proyecto sentar las bases y realizar un esbozo o esqueleto inicial de un entorno de trabajo de estas características.

La característica principal de la que se ha querido dotar el entorno consiste en que la interacción del usuario con el entorno sea mediante la ejecución y operación de transacciones. Una transacción, que comprende un conjunto de operaciones específicas sobre el sistema, lleva siempre asociada un código de transacción. El usuario, para utilizar el entorno de simulación, ejecutará una o varias de las transacciones que sean adecuadas a las operaciones que desea realizar.

La persistencia de los datos y parámetros que se manejan en la simulación debe ser superior, cuando se necesite, a una sesión de trabajo. Esto significa, que todas las configuraciones y parametrizaciones, simulaciones, cálculos, etc., que el usuario realice deberían estar disponibles para que el usuario las pueda reutilizar si así lo desea en nuevas sesiones de trabajo. Esta faceta, se soluciona utilizando una base de datos o algún mecanismo de almacenamiento similar.

Otra característica muy importante en la que se ha querido basar el entorno de simulación, es una gestión sistemática de los datos que los divide en tres clases básicas:

- Datos de configuración: Datos que gobiernan el funcionamiento y las posibilidades que ofrece una herramienta. Estos datos deberían estar predefinidos para los usuarios y no deberían en principio modificarlos, sino que su gestión corresponde a los desarrolladores del sistema.
- Datos maestros: Hay ciertos datos o conjuntos de datos que son utilizados reiteradamente en las distintas operaciones que se realizan sobre el entorno. Para evitar que el usuario tenga que introducir repetidamente los mismos datos, e identificar estos patrones de datos, estos datos se gestionan como datos maestros. Esto implica, que a un dato o conjunto de datos se le asigna un tipo de dato maestro y un código de dato maestro. El usuario puede referirse a él siempre que lo necesite utilizando ese código.

- Datos transaccionales: Los datos que no son ni de configuración ni maestros, son por exclusión datos transaccionales. Todos los datos que el usuario registra en el sistema para reflejar procesos o parámetros son datos transaccionales. Estos pueden comprender datos de cualquier tipo, incluidos códigos de datos maestros o de datos de configuración. Los usuarios registrarán en el sistema datos transaccionales para ajustar el funcionamiento de la simulación a sus requerimientos.

Para apuntalar una buena gestión y control de los datos otra faceta muy importante a integrar en el entorno de simulación es la incorporación de un diccionario de datos. Un diccionario es un repositorio de tipo de datos que permite contrastar los valores que toman una serie de datos frente a unas especificaciones fijas. De esta forma, se puede garantizar que datos que deban tener características similares, a base de contrastarlos con el tipo de dato apropiado del diccionario de datos, las tengan.

Con respecto al entorno gráfico una prestación muy importante es la operación mediante "modos". Cada sesión de usuario permite la utilización de uno o varios modos (se pueden entender como ventanas) que permiten ejecutar transacciones independientes entre sí. Esto, hará más eficiente la interacción del usuario con la herramienta, permitiéndole por ejemplo, que simultáneamente a la selección de un dato maestro en una transacción, pueda consultar en un modo distinto los valores asociados a este dato maestro.

Para enfrentarse a un entorno complejo con multitud de componentes, relacionados entre sí, y cuya complejidad crece constantemente con el paso del tiempo, se requiere una estrategia de gestión de los componentes que se van desarrollando. Esta gestión se basará en conceptos como agrupaciones, versiones, parches, transportes etc. Igualmente, disponer de varios entornos, de desarrollo, de pruebas, productivo es también muy importante a la hora de gestionar un entorno de simulación complejo y cambiante.

Por último, el entorno de simulación y todas las operaciones que los usuarios realizan sobre él, deben tener un "look and feel" o aspecto similar y reconocible, para evitar que el usuario tenga que estar aprendiendo constantemente como utilizar las distintas funcionalidades que se ofrezcan.

Hay muchos más aspectos que sería muy interesante añadir a la herramienta de simulación, pero debido a limitaciones de capacidad, estos se dejarán especificados como posibles líneas de mejora, que se pueden abordar en el futuro.

3.2. Implementación.

Se ha decidido que el lenguaje de programación en que se va a construir el modelo del entorno de simulación será Matlab. Se ha elegido este lenguaje porque, en un principio, está especialmente adaptado para trabajar con señales y realizar cálculos complejos. Sin embargo, a la hora de utilizarlo ha resultado estar especialmente mal equipado para el resto de tareas que son necesarias para implementar el entorno de simulación según se ha descrito. Sería bastante conveniente sustituirlo gradualmente por otro lenguaje más apropiado para el núcleo de

prestaciones de la simulación y codificar en Matlab solo aquellas partes directamente relacionadas con el procesado de señales y quizás la muestra de gráficos.[1]

La base de datos necesaria para la persistencia de los datos, se ha simulado mediante ficheros locales de Matlab, a su vez basados en la estructura de Matlab de mapas de datos, que es la única forma que permite Matlab de organizar por clave una serie de registros. Como Matlab no está especialmente preparado para este tipo de gestión de datos, se ha requerido construir por debajo una capa de software que abstraiga este problema. Para poder trabajar de forma distribuida y mejorar la eficiencia del sistema sería conveniente reemplazar de forma transparente el almacenamiento en ficheros por almacenamiento en una base de datos.

Debido a las limitaciones inherentes a construir un prototipo, el entorno de simulación se ha construido para ejecutarse localmente en un ordenador, para que el entorno de simulación se pueda utilizar a un potencial razonable, más allá de un prototipo, sería necesario cambiar la arquitectura a un modelo cliente servidor, de forma que la carga fuerte de procesado se ejecutase en un servidor de altas prestaciones y el acceso de los usuarios fuese mediante un cliente ligero que podría ser incluso un navegador.

Se ha pretendido que el software desarrollado, esté construido basado en el paradigma de orientación a objetos; esto es así, porque se considera la forma más apropiada de gestionar una cantidad grande de funcionalidad; de forma que, facilite su escalado y mantenimiento. Sin embargo, aunque la versión de Matlab utilizada se supone que incorpora la orientación a objetos, alguna de sus características no son las que se esperarían de un lenguaje orientado a objetos y ha sido necesario desarrollar una "capa" de código que recubra la implementación de las clases para hacer esto transparente a la hora de desarrollar.

Con respecto a la gestión de los desarrollos, no tenía mucho sentido diferenciar entornos en una fase en la que el entorno de simulación está en "prototipo", y basta con organizarlos todos bajo la agrupación de versión 0. Cuando se empiece a utilizar y según su complejidad vaya creciendo, sí que será necesario diferenciar entornos de trabajo y desarrollar algún tipo de nomenclatura y herramienta que permita gestionar las distintas versiones que se vayan realizando de los componentes. Por ahora, la única nomenclatura que se ha utilizado (aparte de la apropiada a cada clase de objeto de desarrollo) es mantener prefijos relativos al bloque en el que se puede encuadrar cada objeto.

En el siguiente capítulo se detallarán los componentes desarrollados con arreglo a las líneas que aquí se han expuesto.

[1] Cualquier lenguaje de los ampliamente utilizados para el desarrollo de aplicaciones sería más versátil que el propio Matlab, ya que suelen llevar incorporadas una cantidad importante de funciones que en Matlab es necesario desarrollar. Igualmente, la orientación a objetos en Matlab no se ha incorporado de forma estándar, siendo bastantes de sus propiedades distintas de sus equivalentes en otros lenguajes. Apuntar que el propio SAP tiene un lenguaje propio, el ABAP, que incorpora implícitamente muchas de las funcionalidades que ha sido necesario desarrollar.

CAPíTULO 4. DESARROLLO DEL ENTORNO DE SIMULACIÓN.

En este capítulo se explican y detallan los distintos objetos de desarrollo que se han creado para sustentar el funcionamiento del entorno de simulación. No se pretende descender al detalle línea a línea del código pero si dar las suficientes explicaciones como para permitir mantenerlo y ampliarlo.

La mayoría de los objetos de desarrollo respetan una nomenclatura que ayuda a la legibilidad del código y a su comprensión.[ii]

4.1. EL CÓDIGO DE RETORNO.

Normalmente cada vez que se ejecuta una instrucción o un módulo de instrucciones es conveniente evaluar el resultado de la ejecución (satisfactorio, insatisfactorio, otras posibilidades,…). En muchos lenguajes, hay variables que de forma implícita proveen esta información en forma de "código de retorno" cuyo valor se puede evaluar para conocer cuál ha sido el resultado del proceso.

Esto no existe en Matlab. Como alternativa, cuando se ejecutan funciones de Matlab, sería posible siempre declarar como parámetro explícito de salida, una variable que gestionase esta información. Aunque inicialmente esta faceta se implementó así, ello no contribuía para nada a la claridad, teniendo que especificar la misma variable cada vez que se llamaba a una función.

Para evitar esto, y mejorar en general la legibilidad del código; se utiliza, en vez de una variable declarada explícitamente en cada interface de función, una variable de sistema. Siempre que se quiera evaluar el procesado de una función bastará con consultar el valor que toma esta variable justo después de la ejecución. Al codificar esta función habrá que estar atento a informar esta variable convenientemente.

Como regla general, se trata de una variable con un número entero positivo, que debería tomar el valor 0 cuando la ejecución ha sido correcta y cualquier otro valor (casi siempre 4) cuando se ha producido una incidencia en la ejecución.

Como Matlab no tiene incorporadas por defecto variables de sistema, se ha creado una clase "blx_cl_syvar" a la que se puede acceder en modo singleton y cuyos atributos recogerán las variables de sistema. Entre estos atributos está el código de retorno (rc); para el cual se han creado dos funciones de nomenclatura abreviada (que llamaremos macros y cuyo sentido se explicará más adelante) que permiten gestionar su valor. La función src (set return code) permite informarlo y la función grc (get return code) recuperar su valor.

Un motivo recurrente en el código por lo tanto será la continua gestión mediante estas funciones del valor de esta variable. Aparte de mejorar, la legibilidad del código y evitar el trabajo repetitivo que sería necesario de otra manera; permite depurar muy bien cualquier tipo de error, simplemente vigilando cuando esta variable toma un valor distinto de cero.

4.2. EL DICCIONARIO DE DATOS.

Para controlar la información que se maneja en la simulación es necesario garantizar cierta uniformidad en las características de los datos que representen información semejante.

Por ejemplo, una variable que almacenase la velocidad de un blanco, no debería poder tener caracteres ya que no podría operarse con otra variable de tipo velocidad que tuviese valores numéricos.

Con el fin de poder ejercer este control, se ha desarrollado una clase, blx_cl_dadom, que permite contrastar y verificar los valores de las variables. Es importante aclarar, que se trata de un diccionario sintáctico no semántico, i.e., verifica la forma del dato, no el significado del dato. Una mejora a las prestaciones del entorno, sería la incorporación de una capa de diccionario semántico sobre la existente de diccionario sintáctico.

El diccionario de datos que se ha creado, utiliza como componentes básicos unas entidades denominadas dominios. Un dominio, es una secuencia de caracteres que lleva asociadas una serie de características o restricciones sobre los valores de una variable. En tiempo de ejecución, se pueden utilizar las prestaciones del diccionario de datos para verificar que el valor de una variable se ajuste a las especificaciones de un determinado dominio.

4.2.1. VERIFICACIÓN DE DOMINIO.

Utilizar el diccionario de datos para verificar un dominio es tan sencillo como instanciar el gestor de dominios (clase blx_cl_dadom) y utilizar el siguiente método:

```
blx_mt_chk_field(blx_ob_dadom,blx_xx_fdval,blx_st_donam)
```

El primer parámetro, será la instancia del gestor de dominios, el segundo la variable cuyo contenido se quiere verificar, y el tercero el nombre del dominio contra el cual se quiere contrastar el valor de la variable.

Esta llamada se ha encapsulado con una “macro” y se comentará en su correspondiente apartado.

4.2.2. DEFINICIÓN Y ESTRUCTURA DE LOS DOMINIOS.

La definición de las características de los dominios requiere de una estructuración y organización que simplifique y faciliten esta tarea. Por ello, para asegurar una estructura en la definición de las características de los dominios, se ha optado por organizar su definición según un sistema jerárquico de composición. Esto es, se parte de unas características elementales que se combinan en distintos grados de complejidad para definir las características de los dominios.

4.2.2. DOMINIOS ELEMENTALES.

Toda la caracterización de los dominios, se basa en unas entidades denominadas dominios elementales. Estos dominios son la unidad mínima de definición de características y no pueden ser utilizados directamente para contrastar valores de variables, sino que son conjuntos de estos dominios elementales los que quedan asociados a los dominios que finalmente se utilizarán.

Los dominios elementales se definen en el método “blx_mt_load_elemd_map” y consisten en una secuencia de caracteres que los identifica y un método asociado a esta secuencia que contiene el código que realizará las verificaciones asociadas a este dominio elemental. Para la nomenclatura de los dominios elementales se utiliza la siguiente convención: dos caracteres iniciales para el tipo de dato, un guion bajo y luego una secuencia de caracteres para identificar el dominio elemental.

La verificación de las características de un dominio elemental puede hacerse depender de parámetros, por lo tanto en la definición del dominio elemental, además del nombre del dominio elemental y el nombre del método que realiza la verificación se especifica una clave que se utiliza para verificar el valor de estos parámetros. Para definir estas claves de verificación de valores de parámetros junto con los métodos que realizan esta verificación se utiliza el método "blx_mt_load_edomp_map".

4.2.3. DOMINIOS SIMPLES.

Este es el tipo de dominio más sencillo que se puede utilizar para verificar valores de variables, consiste básicamente en la especificación de una serie de dominios elementales y sus parámetros cuya verificación garantice que el dato se ajusta a lo representado por el dominio simple.

La definición de los dominios simples se realiza en el método "blx_mt_load_simpd_map" que a su vez, debido al número elevado de definiciones, se ha separado en métodos específicos por módulo con la nomenclatura "blx_mt_load_xxx_simpd_map"; donde xxx representa la serie de caracteres que identifican un bloque de funcionalidad. Por ejemplo, en el método "blx_mt_load_bl1_simpd_map" se definirán los dominios simples asociados en el módulo de blancos.

A partir de la especificación de los dominios simples se pueden especificar otra serie de tipos de dominios más complejos, que estarán compuestos a base de dominios simples.

4.2.3. DOMINIOS ESTRUCTURADOS.

Estos dominios se utilizan para contrastar variables estructuradas. En la especificación de un dominio estructurado, junto al nombre del dominio se específica el nombre de los campos que contiene la estructura, el dominio simple al cuál debe estar sujeto cada campo de la estructura, así como si alguno de los campos de esta estructura es opcional y solo debe verificarse cuando exista.

Al igual que para los dominios simples, la definición de los dominios estructurados, se realiza en el método "blx_mt_load_strud_map", que a su vez está compuesto de métodos para la definición específica de los dominios asociados a cada bloque de funcionalidad con una nomenclatura del tipo "blx_mt_load_xxx_strud_map".

4.2.4. DOMINIOS MAPA.

Estos dominios se utilizan para verificar variables de tipo mapa. Los mapas son referencias a la clase de Matlab "containers.Map" que se utiliza ampliamente en el entorno de simulación ya que es lo más parecido que Matlab proporciona a una variable interna que permita acceder a sus registros mediante clave. En la especificación de un dominio mapeado, se especifican tres dominios simples, uno que aplique a toda la variable, otro que aplique a las claves del mapa, y otro a los registros del mapa.

La definición de los dominios mapa se realiza como en los casos anteriores en un método específico "blx_mt_load_cmapd_map" que a su vez está compuesto de métodos específicos para cada módulo de funcionalidad con la nomenclatura "blx_mt_load_xxx_cmapd_map".

4.2.5. DOMINIOS CELDA.

Estos dominios se utilizan para verificar variables de tipo celda en las que se desea que ciertos conjuntos de celdas tengan características específicas. Al especificar estos dominios se puede especificar una serie de dominios elementales que apliquen a todas las celdas así como dominios elementales que apliquen a rangos de celdas.

Como ocurre con el resto de tipos de dominios, la definición se realiza en el método "blx_mt_load_celld_map" que está diferenciado en métodos específicos por módulo con la nomenclatura "blx_mt_load_xxx_celld_map".

4.2.6. VALORES FIJOS DE DOMINIOS.

Una característica que se puede asociar a los dominios simples es restringirlos a una serie de valores fijos.

Estos valores fijos se especifican en el método "blx_mt_load_fixdo_map" que, como en el resto de los casos, está diferenciado en métodos por bloque de funcionalidad del tipo "blx_mt_load_xxx_fixdo_map".

Cuando se verifique un dominio simple, si tiene asociados valores fijos, además de contrastar el contenido de una variable contra lo especificado en una serie de dominios elementales se comprobará si el valor es uno de los especificados en sus dominios fijos.

Existe la posibilidad de recuperar los valores fijos asociados a un dominio elemental, a partir de una instancia del gestor de dominios, utilizando los métodos:

```
blx_ce_doval = blx_mt_get_fixva(blx_ob_dadom,blx_st_donam)
blx_sa_doval = blx_mt_get_fixda(blx_ob_dadom,blx_st_donam)
```

Uno de los métodos devuelve solo los valores fijos, mientras que el otro devuelve además una descripción asociada.

4.2.6. VERIFICACIÓN DE CONSISTENCIA DE DEFINICIONES DE DOMINIOS.

Para mejorar la eficiencia en el uso del diccionario de datos, el acceso a la clase que lo gestiona debe realizarse en modo singleton. Esto implica, que la clase solo se instancia una vez y luego se accede a esta instancia desde distintos puntos del código cuando sea necesario. Esto evita tener que repetir todo el procesado de carga de los datos de los distintos dominios que se realiza cada vez que se instancia la clase.

Para acceder a la instancia de la clase en modo factoría, se ha desarrollado el método "blx_mt_get_data_dom". Además casi toda la funcionalidad de esta clase está encapsulada en "macros" para que su llamada sea relativamente sencilla.

Aparte de cargar los datos de los dominios cada vez que se instancia la clase, también se realiza una verificación de la corrección de los datos. Esto consume recursos, y es posible activar y desactivar esta funcionalidad si el rendimiento así lo requiere. Por ejemplo, cada vez que se realicen cambios en las definiciones de los dominios se puede activar esta funcionalidad para que se verifique la consistencia de los cambios y luego desactivarla una vez que la

consistencia está garantizada. Esta funcionalidad se gobierna mediante el valor del atributo "blx_st_chkdo" de la clase "blx_cl_dadom".

4.3. LOS MAPAS DE DATOS.

Parte de la información que se maneja en el entorno de una simulación requiere de una persistencia; con esto se quiere decir que datos que se hayan introducido, registrado, calculado, etc. en una sesión de trabajo deben estar disponibles en sesiones subsiguientes. Por ejemplo, si durante una sesión de trabajo se han estado configurando las características de una antena, tiene sentido poder volver a referenciar estos datos registrados en una sesión posterior sin necesidad de volver a registrarlos.

A parte de los datos maestros y transaccionales, que los usuarios del entorno de simulación registren, es necesario para su funcionamiento disponer en cada sesión de una serie de datos que gobiernen la configuración y prestaciones del sistema.

Para dar solución a estas necesidades se ha creado un gestor de mapas de datos. Esta herramienta, pretende sustituir a lo que en otros entornos sería una base de datos relacional. Se considera que el funcionamiento del gestor de mapas de datos debe ser perfeccionado en el futuro sustituyendo el actual funcionamiento por la utilización de una base de datos, quedando la funcionalidad actual como un envoltorio de forma que el cambio fuese lo más transparente y sencillo posible.

Según se ha desarrollado el gestor de mapas de datos, se basa en el almacenamiento de ficheros .mat (ficheros con variables de Matlab y sus valores) de una serie de mapas de datos. Además, estos mapas de datos tienen asociados unos metadatos, que asemejan su funcionalidad a la de una tabla de base de datos relacional.

El gestor de mapas manejará unas entidades denominadas "mapas de datos" los cuales llevarán asociados a su nombre, una serie de metadatos, y con los cuales se podrá interactuar como si de una tabla de base de datos se tratará: leer registros, modificarlos, eliminarlos. Para garantizar la persistencia, toda esta información se guardará en ficheros cada vez que se finalice una sesión de trabajo.

4.3.1. DEFINICIÓN DE MAPAS DE DATOS.

Un mapa de datos consistirá pues en una serie de registros a los que se accede por una clave, todos los registros tendrán en común una serie de campos con las mismas características. Se puede considerar que la definición de un mapa de datos tiene dos partes diferenciadas, por un lado, la definición de los datos relativos a todo el mapa o datos de cabecera y por otro lado los datos relativos a los campos del mapa. Además algunos de los campos (los que se consideren de configuración), vendrán con una serie de datos predeterminados o "de fábrica" que debe ser posible cargar la primera vez que se utilicen o siempre que sea necesario.

4.3.1.1. Definición de metadatos de cabecera de un mapa.

Para definir un mapa hay que especificar los siguientes datos:

- **Nombre del mapa:** Cada mapa debe tener un nombre único que se ajuste a unas convenciones de nomenclatura (Ver nota).
- **Descripción:** Una descripción corta (menos de 50 caracteres) de la función del mapa.
- **Tipo de mapa:** Cada mapa debe ser clasificado según el tipo de datos que vaya a gestionar:

C -> Datos de configuración.
M -> Datos maestros.
T -> Datos transaccionales.

Esta clasificación tiene consecuencias sobre la utilización que se puede hacer del mapa.

- **Fichero:** Nombre del fichero donde se almacenará el contenido del mapa. Como regla general el nombre del mapa y el nombre del fichero suelen coincidir.
- **Nombre del campo clave:** Todo mapa debe tener un campo clave, y se debe indicar su nombre.
- **Dominio del campo clave:** La clave del mapa debe estar sujeta a unas restricciones sobre sus posibles valores. Estas restricciones se aplican asociando el campo clave a un dominio del diccionario de datos.
- **Mapa de verificación:** En una base de datos relacional, se puede forzar a que ciertos campos solo puedan tomar valores que existan como claves de un segundo mapa. Si este es el caso, se puede indicar el nombre del mapa contra el que se debe verificar los valores del campo clave.
- **Campo de texto:** Algunos mapas tienen un campo que describe específicamente cada registro del mapa. Este campo se denomina campo de texto del mapa y suele ser una descripción del valor del campo clave del mapa. Esto se utiliza para ayudar a seleccionar registros en una tabla. Si un mapa tiene uno de sus campos funcionando como campo de texto se debe indicar aquí.
- **Método de inicialización:** Los mapas con datos de configuración y algunos mapas con datos maestros tienen datos predefinidos que se deben poder cargar siempre que sea necesario. Si un mapa tiene estos datos, se debe indicar aquí el nombre del método que realiza la carga de estos datos.
- **Método de metadatos de campos del mapa:** Los mapas aparte del campo clave tienen en cada registro una serie de campos con unas características asociadas. Aquí se especifica cuál es el método que se debe utilizar para obtener esta información.

Todos estos metadatos de "cabecera" de un mapa se indican en el método "blx_mt_load_metadata_map_header". Como ocurre en muchos otros casos, debido al volumen de datos y para clasificar su información este método se subdivide en métodos específicos por bloque de funcionalidad donde se definen los metadatos de los mapas asociados con cada bloque.

4.3.1.2. Definición de metadatos de los campos de un mapa.

Aparte de los metadatos de cabecera, para definir un mapa hace falta definir las características de sus campos, ello se hace especificando los siguientes datos:

- **Nombre del campo**: El campo debe tener un nombre para poder referirse a él.
- **Descripción del campo:** Una descripción corta (menos de 50 caracteres) de la información que está prevista que contenga el campo.
- **Dominio del campo:** Para especificar las características que deben tener los valores de los registros para este campo se utiliza un dominio del diccionario de datos.

- **Mapa de verificación:** Si se desea que el campo solo pueda tener unos valores específicos que coincidan con los campos claves de otro mapa, ese otro mapa se especifica aquí.

Todos estos datos se definen en el método especificado en los metadatos de cabecera del mapa respectivo.

4.3.2. OPERACIONES SOBRE MAPAS.

Sobre un mapa es posible realizar todas las operaciones que es normal realizar sobre una tabla de base de datos. A continuación se especifican junto a los correspondientes métodos que hay que utilizar para ejecutarlas. Siempre es necesario partir de una instancia del gestor de mapas.

Hay que tener en cuenta que todas estas operaciones han sido encapsuladas en funciones macro que se explicarán más adelante.

4.3.2.1. Leer registro.

Es posible leer datos de campos de un registro de un mapa llamando al siguiente método:

```
blx_sc_maval = blx_mt_get_map_reg_fields(blx_ob_damap,blx_st_mpnam,blx_xx_mpkey,blx_ce_mafld)
```

Este método devolverá un código de retorno cero si ha encontrado un registro y un código de retorno distinto de cero en otro caso.

Los parámetros tienen la siguiente función y significado:

- 'blx_sc_maval'. Si se ha encontrado un registro, esta estructura tendrá tantos campos como se hayan requerido con el correspondiente valor que tengan en el registro.
- 'blx_ob_damap'. Esta es la referencia a la instancia del gestor de mapas.
- 'blx_st_mpnam'. Nombre del mapa en el que se busca el registro.
- 'blx_xx_mpkey'. Clave del registro que se desea buscar.
- 'blx_ce_mafld'. Celda con los nombre de los campos del registro que se desea recuperar. Si no se especifica ninguno, se recuperarán todos los campos del registro.

4.3.2.2. Insertar modificar/registro.

Otra de las operaciones más comunes sobre un mapa es añadir un registro no existente o modificar uno que ya exista. En el caso de que ya exista el registro se modificará según los valores indicados. Para realizar esta operación se llamará al siguiente método:

```
blx_mt_set_map_reg_fields(blx_ob_damap,blx_st_mpnam,blx_xx_mpkey,blx_sc_maval)
```

Este método devolverá un código de retorno cero si fue capaz de crear/modificar el registro y cero en caso contrario.

Los parámetros tienen la siguiente función y significado:

- 'blx_ob_damap'. Esta es la referencia a la instancia del gestor de mapas.
- 'blx_st_mpnam'. Nombre del mapa en el que se desea actualizar el registro.
- 'blx_xx_mpkey'. Clave del registro que se desea crear/modificar.
- 'blx_sc_maval'. Estructura con los campos y sus valores que se desean actualizar en el registro. Si se desea crear un registro nuevo, en esta estructura deberán de ir todos los campos necesarios, mientras que si se desea actualizar uno ya existente solo son necesarios los campos que se deseen modificar con sus valores.

4.3.2.3. Borrar registro.

Se puede borrar un registro de un mapa a partir de su clave utilizando el siguiente método:

```
blx_mt_del_map_reg(blx_ob_damap,blx_st_mpnam,blx_xx_mpkey )
```

De nuevo el código de retorno será cero si el registro se ha borrado con éxito y distinto de cero si ha ocurrido cualquier problema al intentar borrarlo.

Los parámetros tienen la siguiente función y significado:

- 'blx_ob_damap'. Esta es la referencia a la instancia del gestor de mapas.
- 'blx_st_mpnam'. Nombre del mapa en el que se desea borrar el registro.
- 'blx_xx_mpkey'. Clave del registro que se desea borrar.

4.3.2.4. Borrar todos los registros.

Para borrar todos los registros de un mapa sin tener que borrar uno por uno se puede utilizar el siguiente método:

```
blx_mt_del_map_regs(blx_ob_damap,blx_st_mpnam)
```

El código de retorno será cero si los registros del mapa se han borrado sin problemas y distinto de cero para reflejar cualquier incidencia.

Los parámetros tienen la siguiente función y significado:

- 'blx_ob_damap'. Esta es la referencia a la instancia del gestor de mapas.
- 'blx_st_mpnam'. Nombre del mapa en el que se desea borrar los registros.

4.3.2.5. Resetear el mapa a valores iniciales.

Cuando se trate de un mapa de configuración o de datos maestros y se desee volver a la información predefinida, se puede utilizar el siguiente método:

```
blx_mt_reset_map_regs(blx_ob_damap,blx_st_mpnam)
```

El código de retorno será cero si el mapa se ha devuelto a su estado inicial y distinto de cero ante cualquier problema.

Los parámetros tienen la siguiente función y significado:

- 'blx_ob_damap'. Esta es la referencia a la instancia del gestor de mapas.
- 'blx_st_mpnam'. Nombre del mapa cuyos valores se desean reiniciar.

4.3.2.6. Recuperar claves del mapa.

Otra operación útil es obtener las claves de todos los registros cargados en un mapa, para ello, se utiliza el siguiente método:

```
[blx_ce_mpkey,blx_ce_mpktx] = blx_mt_get_map_keys(blx_ob_damap,blx_st_mpnam)
```

El código de retorno será cero si se han recuperado las claves sin problema y distinto en cualquier otro caso.

Los parámetros tienen la siguiente función y significado:

- 'blx_ce_mpkey'. Celdas con las claves del mapa.
- 'blx_ce_mpktx'. Si el mapa tiene asociado campo de texto, en este campo vendrán las celdas con los valores de las descripciones asociadas a cada clave.
- 'blx_ob_damap'. Esta es la referencia a la instancia del gestor de mapas.
- 'blx_st_mpnam'. Nombre del mapa cuyos valores se desean reiniciar.

4.3.2.7. Mostrar datos del mapa.

Esta operación permite mostrar de manera sencilla los datos que están cargados en un mapa. Es necesario llamar a este método:

```
blx_mt_dsp_map(blx_ob_damap,blx_st_mpnam)
```

Se mostrarán los datos listados en la pantalla de comandos. El código de retorno será cero si se ha producido cualquier incidencia al mostrarlos.

Los parámetros tienen la siguiente función y significado:

- 'blx_ob_damap'. Esta es la referencia a la instancia del gestor de mapas.
- 'blx_st_mpnam'. Nombre del mapa cuyos valores se desea mostrar.

4.3.3. ALMACENAMIENTO DE LOS MAPAS DE DATOS.

Para que los datos gestionados mediante el gestor de mapas de datos, tengan persistencia entre sesión y sesión es necesario almacenarlos. El almacenamiento se realiza actualmente de forma rudimentaria en ficheros de variables de Matlab, siendo las variables que se almacenan en estos ficheros los propios mapas con sus datos.

Los ficheros se almacenan en la carpeta “data_maps” existiendo una subdivisión en tres carpetas que corresponden a los tres tipos de mapas. Cada mapa se almacenará en un fichero independiente cuyo nombre se indica en la configuración de los metadatos de cabecera del propio mapa.

Los siguientes métodos, permiten respectivamente, cargar un mapa desde fichero o guardar un mapa en fichero.

```
blx_mt_load_map(blx_ob_damap,blx_st_mpnam)
blx_mt_save_map(blx_ob_damap,blx_st_mpnam)
```

Además para guardar todos los mapas se puede utilizar el método:

```
blx_mt_post_maps(blx_ob_damap)
```

Este método se llama por defecto siempre que se finaliza una sesión con lo que se garantiza una persistencia de los datos. Esto implica, que para reinicializar un mapa desde su estado inicial (datos predefinidos o vacio), es necesario llamar implícitamente al método de “reseteo” o bien borrar manualmente el archivo correspondiente al mapa antes de comenzar una sesión.

4.4. CLASES TIPIFICADAS.

Un problema que presenta Matlab es que no permite caracterizar ni restringir a priori los valores de los atributos de una clase. Otros lenguajes con diccionario de datos permiten referirse a él a la hora de definir atributos, parámetros o variables.

Adicionalmente a este problema se presenta otro: Cuando se utiliza la técnica de herencia para derivar la funcionalidad de una clase de otra, los atributos que se declaran como públicos o protegidos, no se diferencian entre hijos y padres cuando son referencias a objetos de tipo “handle”[2] sino que se comparten.

Para enfrentarse a estos problemas, se ha creado una clase “raíz” “blx_cl_rooto”, que añade la funcionalidad de tipificación de sus atributos, y permite la herencia de las clases “handle” sin que haya conflicto en los atributos.

La clase “blx_cl_rooto” es una clase de tipo handle por lo que hereda a su vez de la clase base de Matlab “handle”. Cuando se desee que una clase incorpore la funcionalidad mencionada no hay más que indicar que es heredera de la clase “blx_cl_rooto”.

[2] Matlab distingue entre dos tipos de clases, las llamadas clases “handle” y las llamadas clases “value”. Básicamente las clases “handle” se almacenan una vez en memoria y se manejan cuantas referencias a ellas sean necesarias, mientras que las clases “value” pueden tener tantas instancias y sets de datos distintos en memoria como se desee. Adicionalmente, las clases “handle” heredan una serie de métodos y características que les permite utilizar toda la potencia de la orientación a objetos. La herencia de clases “handle” no está en mi opinión bien implementada puesto que hace compartir a padres e hijos un mismo espacio de memoria y por tanto las mismas variables.

La tipificación de los atributos de las clases “tipificadas” se hace asignando a cada instancia de clase derivada de la clase “blx_cl_rooto” un número de objeto y guardando por cada número de objeto un mapa con los atributos pertenecientes a ese objeto.

Todas las clases que herederas de la clase “blx_cl_rooto”, disponen del método “blx_mt_set_reg” que permite definir uno a uno los atributos con los que contará la clase. Preferentemente esta definición se debe hacer en el método “blx_mt_init_attr” que tendrán que definir todas las clases herederas. Cuando se define el atributo de una clase se han de indicar los siguientes datos:

- **Nombre**: El atributo debe tener un nombre único ajustado a nomenclatura.
- **Descripción**: Un texto corto que describa la función del atributo.
- **Dominio:** Los valores y características de un atributo se pueden especificar especificando un dominio del diccionario de datos contra el que deban contrastarse.
- **Mapa de verificación:** Si se desea se pueden limitar los valores del atributo a las claves existentes en un mapa del gestor de mapas. Si es así se debe indicar el nombre del mapa.
- **Función de verificación:** Si se desea hacer alguna verificación adicional sobre los valores del atributo cuya complejidad sobrepase a lo que permita un dominio o un mapa de verificación, aquí se puede apuntar a una función que contenga el código que realice estas verificaciones.

4.4.1. OPERACIONES CON ATRIBUTOS DE CLASES TIPIFICADAS.

Para utilizar los atributos de clases tipificadas no se pueden utilizar los métodos válidos para las clases normales, si no que se deben utilizar los métodos habilitados para tal fin. Estos métodos están inmediatamente disponibles en todas las clases herederas de la clase “blx_cl_rooto”.

4.4.1.1. Lectura de valor de atributo.

Para leer el valor de un atributo o de varios se deben utilizar los siguientes métodos:

```
blx_xx_atval = blx_mt_get_atval(blx_ob_rooto,blx_st_atnam)
blx_sc_atval = blx_mt_get_atvls(blx_ob_rooto,blx_ce_atnam)
```

El método “blx_mt_get_atval” permite recuperar el valor de un atributo en específico, indicando su nombre en la variable “blx_st_atnam”.

El método “blx_mt_get_atvls” permite recuperar el valor de varios atributos, indicando su nombre en la variable “blx_ce_atnam”, que consistirá en una serie de celdas con los nombre de los atributos. El método devuelve la variable estructurada “blx_sc_atval” que tendrá un campo por atributo requerido.

4.4.1.2. Actualización de valor de atributos.

Para actualizar el valor de un atributo o de varios se han de utilizar los siguientes métodos:

```
blx_mt_set_atval(blx_ob_rooto,blx_st_atnam,blx_xx_atval)
blx_mt_set_atvls(blx_ob_rooto,blx_sc_atval)
```

El método “blx_mt_set_atval” permite informar el valor de un atributo, indicando en la variable “blx_st_atnam” su nombre y en la variable “blx_xx_atval” su valor.

Para informar el valor de varios atributos simultáneamente, se puede utilizar el método “blx_mt_set_atvls”. En la variable estructurada “blx_sc_atval” debe haber un campo nombrado como cada atributo que se desee informar, y que tenga asociado el valor del atributo que se quiera actualizar.

4.5. MACROS.

Para utilizar las funcionalidades del gestor del diccionario de datos y del gestor del mapa de datos, es necesario primero obtener acceso a una instancia de estas clases. Además, por motivos de rendimiento, es recomendable que el acceso sea en modo singleton, es decir, que la clase solo se instancie una vez y que se acceda a esa misma instancia desde varios puntos según se necesite. Esto provoca que la legibilidad del código se vea dificultada por una cantidad importante de redundancia en las instrucciones más comunes.

Para suavizar esta dificultad se han creado unos módulos de funciones específicos, con una nomenclatura muy corta y sencilla que enmascaran y sustituyen la secuencia de instrucciones que serían necesarias en su lugar. Estos módulos de funciones se han denominado macro instrucciones o macros en su forma abreviada por semejanza a técnicas similares utilizadas en otros lenguajes de programación.

4.5.1. MACROS PARA EL GESTOR DE DICCIONARIO DE DATOS.

Para utilizar el diccionario de datos de forma ágil se han creado las siguientes macros, todas ellas se han agrupado de tal forma que comparten el prefijo “mxd”.

```
mxd.cfd(blx_xx_fdval,blx_st_donam)
mxd.ccm(i_st_clnam,i_st_mtnam)
l_ce_doval = mxd.gdv(i_st_donam)
l_sa_doval = mxd.gdd(i_st_donam)
```

- “mxd.cfd” -> Permite verificar una variable contra un dominio.
- “mxd.ccm”-> Permite verificar la existencia de un método en una clase.
- “mxd.gdv” -> Recupera los valores fijos de un dominio.
- “mxd.gdd” -> Recupera los valores fijos de un dominio junto con sus descripciones.

4.5.2. MACROS PARA EL GESTOR DE MAPAS DE DATOS.

Para utilizar el gestor de mapas de forma sencilla, se proveen las siguientes macros, todas ellas comparten el prefijo “mxm”.

```
mxm.cmk(i_st_mpnam,i_xx_mpkey)
mxm.cmn(i_st_mpnam)
l_xx_fdval = mxm.gfd(i_st_mpnam,i_xx_mpkey,i_st_fdnam)
l_sc_maval = mxm.gfs(i_st_mpnam,i_xx_mpkey,i_ce_fdnam)
l_ce_mpkey = mxm.gmk(i_st_mpnam)
l_ce_fdnam = mxm.gfn(i_st_mpnam)
mxm.sfs(i_st_mpnam,i_xx_mpkey,i_sc_maval)
mxm.sfd(i_st_mpnam,i_xx_mpkey,i_st_fdnam,i_xx_fdval)
mxm.dmr(i_st_mpnam,i_ce_mpkey)
```

- “mxm.cmk” → Permite verificar que exista un registro en un mapa con una clave específica.
- “mxm.cmn” -> Permite verificar que exista un mapa con un nombre específico.
- “mxm.gfd” -> Permite recuperar el valor de un campo de un registro.
- “mxm.gfs” -> Permite recuperar el valor de varios campos de un registro.
- “mxm.gmk” -> Permite recuperar las claves de los registros existentes en un mapa.
- “mxm.gfn” -> Permite recuperar los nombres de los campos de un mapa.
- “mxm.sfd” -> Permite actualizar el valor de un campo de un registro.
- “mxm.sfs” -> Permite actualizar el valor de varios campos de un registro.
- “mxm.dmr” -> Permite eliminar varios registros de un mapa.

4.5.3. MACROS GENERALES.

Aparte del uso del gestor del diccionario de datos y del gestor de mapas de datos, existen más conjuntos de instrucciones cuya declaración conviene abreviar mediante el uso de macros. Estas se asocian al prefijo “mxg”.

- “mxx.moc” -> Esta macro permite copiar de una variable estructurada a otra aquellos campos que se llamen igual, si en la variable estructurada destino no existen los campos de la variable estructurada origen, estos se crean con la copia (“moc” pretende ser abreviatura de “move-corresponding”).

4.6. VARIABLES DE SISTEMA.

Existe cierta información que es útil para gestionar el entorno de simulación en sí. Esta información debería ser accesible desde cualquier punto. Para poder gestionar esta información se ha creado la clase “blx_cl_syvar” que es accesible desde cualquier punto en modo singleton.

Los atributos de esta clase se utilizarán para guardar esta información de sistema. Por ahora cuenta con la variable “rc” (return code) que como se ha explicado anteriormente guarda la información relativa a la ejecución de cualquier método, función o macro. Cualquier variable que se considere como “de sistema” deberá ser gestionada como atributo de esta clase.

4.7. PARÁMETROS GLOBABLES DE LA SIMULACIÓN.

Igual que existe información considerada de sistema que debe ser accesible desde cualquier punto, también existen una serie de parámetros cuyo significado y uso alcanzan a varios de los componentes de la simulación.

Para gestionar estos parámetros, de forma equivalente a como se hace con las variables de sistema, se ha creado la clase "blx_cl_sgpar" para gestionar estos datos. Igual que ocurre con las variables de sistema, esta clase solo tiene sentido cuando el acceso se realiza en modo singleton y así se puede compartir la información de sus parámetros desde varios puntos distintos del código de la simulación.

4.8. ENTORNO GRÁFICO DE LA SIMULACIÓN.

Uno de los aspectos más importantes de un entorno de simulación es la herramienta que permite al usuario interactuar con él. El usuario debe poder configurar y consultar una cantidad grande de parámetros y observar también un número elevado de datos y resultados. Esta actividad debe ser fácil e intuitiva para el usuario, para ello se le debe ofrecer un entorno gráfico que mantenga un grado de uniformidad lo más elevado posible que le permita aprender a realizar operaciones semejantes con sencillez.

En este espíritu, se ha desarrollado un entorno gráfico de usuario. Este entorno presenta una serie de características que es explican a continuación.

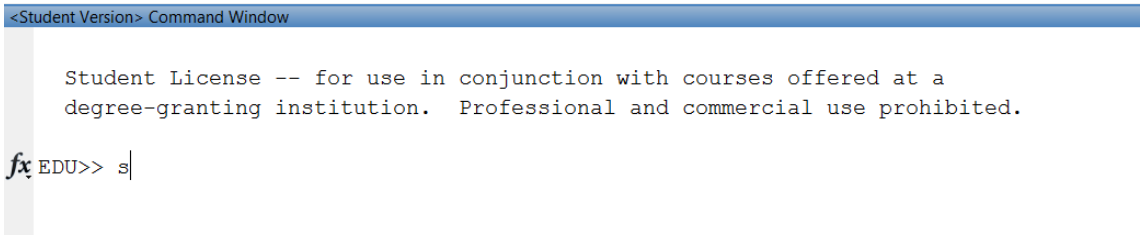
4.8.1. LA SESIÓN DE USUARIO.

Cuando un usuario accede al entorno de simulación, debe identificarse, una vez se ha identificado, se gestionan una serie de datos relacionados con el usuario y su sesión de trabajo. Estos datos permiten luego clasificar las distintas interacciones con el sistema por distintos usuarios y en distintas sesiones.

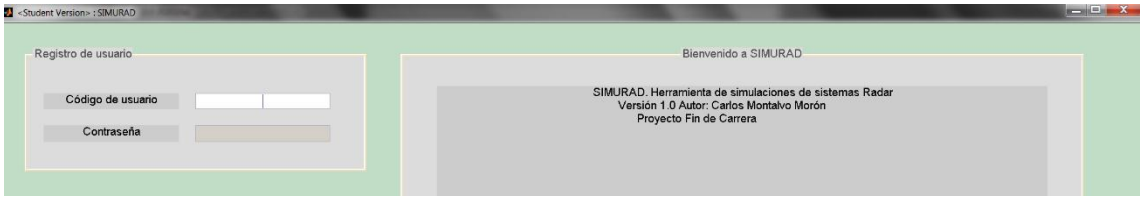
La gestión de los datos de sesión de usuario se ha implementado en la clase "bl0_cl_ses_mng". Esta clase, que funciona en modo singleton, registra entre otras cosas el usuario que está trabajando y la fecha y hora de inicio de la sesión. Es además esta clase la que tiene los métodos que permiten identificarse, iniciar una sesión y finalizarla.

4.8.1.1. Inicio de sesión.

Para iniciar una sesión, el usuario debe teclear en la barra de comandos de Matlab "SIMURAD" o bien la abreviatura "s" (SIMURAD es el nombre provisional dado al entorno de simulación).



Al ejecutar este comando o script, el sistema le mostrará una ventana donde el usuario debe indicar su nombre y una contraseña para poder acceder al entorno de simulación.



4.8.1.2. Maestro de usuarios.

Para gestionar que varios usuarios puedan utilizar simultáneamente el mismo entorno de simulación, y que sea posible distinguir que acciones ha realizado cada usuario, es necesario contar con un registro de usuarios válidos.

Cada usuario estará representado por un código y tendrá asociada una contraseña. Utilizando estos datos es como puede iniciar una sesión en el entorno de simulación.

El maestro de usuarios, o la organización de los datos de usuario, deberá implementarse mediante una serie de mapas de datos relacionados entre sí que permitan gestionar los distintos datos que sea útil registrar por usuario.

Se ha creado inicialmente solo el mapa principal y un mapa auxiliar, pero según se vaya incorporando funcionalidad al entorno, será necesario añadir mapas auxiliares e incrementar el número de campos del mapa principal.

Igualmente, cuando sea posible será necesario habilitar transacciones específicas al entorno gráfico para que un administrador pueda gestionar todos estos datos.

El mapa “bl0_usr1” que tiene como clave el código de usuario, asocia a cada código de usuario una contraseña así como una transacción de inicio e imagen de fondo personalizadas. El mapa “bl0_usr2” que también tiene como clave el código de usuario, guarda para código de usuario datos de control como fecha y hora de creación, usuario creador, etc.

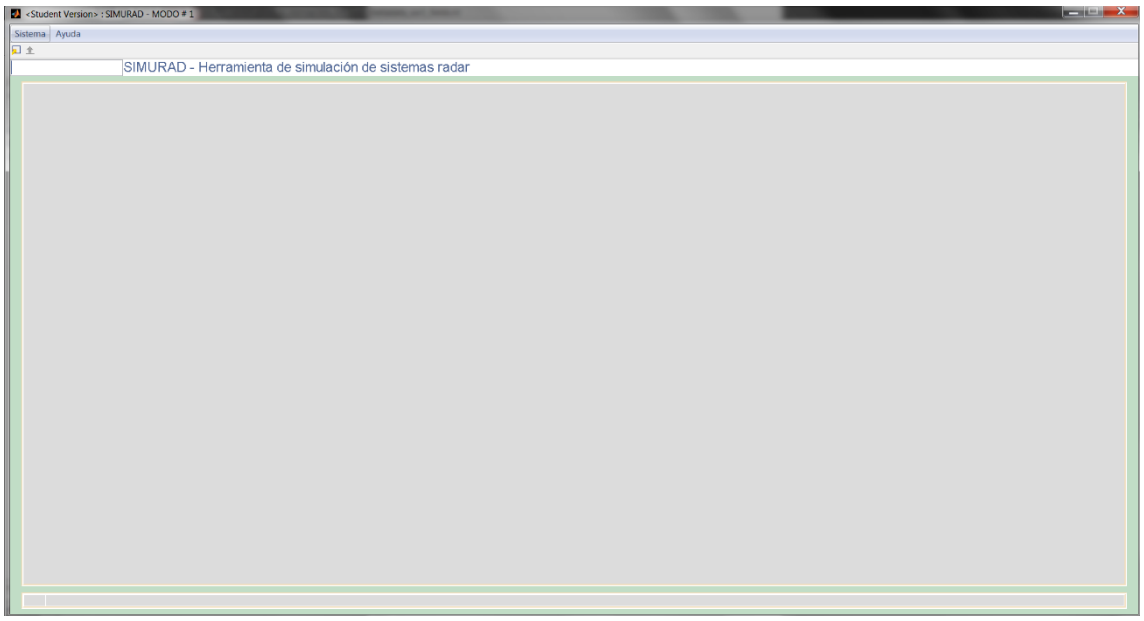
4.8.2. LOS MODOS DE TRABAJO.

Una de las prestaciones que se ha incorporado al entorno gráfico de trabajo, es que se permite trabajar a un usuario en una misma sesión, simultáneamente en varias transacciones.

Una transacción pretende ser la herramienta que pretende realizar una acción específica sobre el entorno. Pues bien, una transacción se ejecuta sobre un modo de trabajo (a partir de ahora “modo” para abreviar). El usuario nada más iniciar una sesión, interactúa con un modo. En este modo inicial, puede ejecutar la transacción que desee, pero también si lo desea puede añadir cuantos modos quiera a su sesión de trabajo para ejecutar y trabajar en varias transacciones simultáneamente.

4.8.2.1. Componentes de un modo.

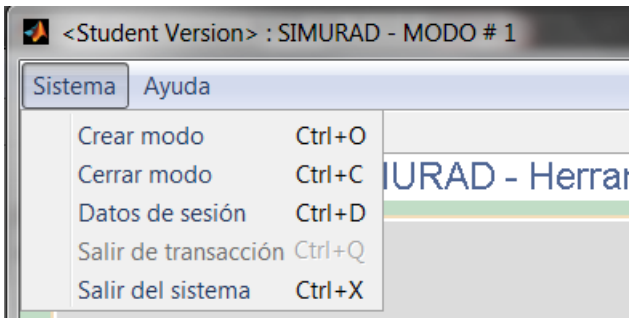
Un modo presenta el siguiente aspecto:



Y en él existen áreas definidas cada una con su función.

Título: En la parte de arriba de cada modo se indica el nombre del entorno de simulación así como el número del modo.

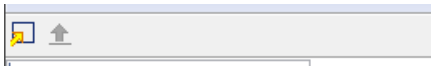
Menús: Justo debajo del título se encuentra la barra de menús. La barra de menús permite al usuario realizar una serie de acciones. Estas acciones son accedidas mediante menús los cuales al desplegarse muestran un listado de ellas entre las que el usuario puede elegir. El número de menús varía con la transacción que se está ejecutando, pero como mínimo siempre habrá dos menús asociados con el propio modo y cuya funcionalidad es independiente de la transacción, el menú de sistema y el menú de ayuda.



Como se ve, además, cada una las acciones de menú pueden llevar asociados un atajo de teclado para que se pueden ejecutar rápidamente y de forma automatizada sin necesidad de navegar por los menús.

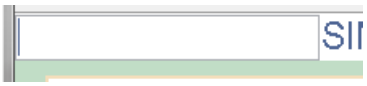
Barra de pulsadores: Otra forma que tiene el usuario de ejecutar acciones dentro de un modo es pulsando sobre los distintos botones que se muestran en la barra de pulsadores. Al

igual que con la barra de menús, los botones que se ofrecen varían de una transacción a otra pero siempre hay un mínimo de dos botones asociados con el propio modo, crear nuevo modo y salir de la transacción que se está ejecutando en el modo.



En este caso como el modo no tiene ninguna transacción ejecutándose solo existen estos dos botones en la barra de pulsadores. Se puede observar que uno de ellos está en gris deshabilitado, esto se debe a que el botón de "salir de la transacción" solo está activo cuando hay una transacción ejecutándose sobre el modo.

Casilla de comandos: El usuario también puede introducir códigos en esta casilla para ejecutar acciones sobre la transacción o sobre el propio modo.



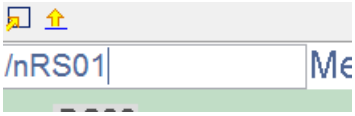
Existen una serie de comandos que introducidos aquí ejecutan acciones independientemente de la transacción que se esté ejecutando sobre el modo:

/n -> Sale de la transacción actual.

/o -> Abre un modo nuevo.

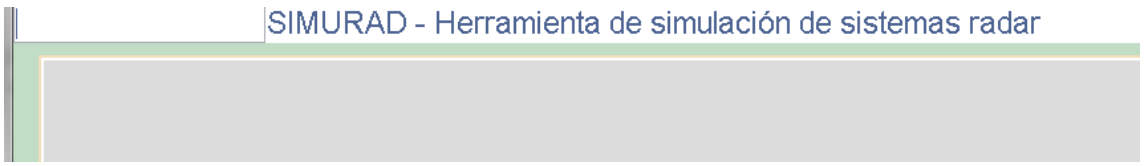
/h -> Activa el modo de depuración.

Después de los comandos /n u /o, o directamente si el modo no tiene transacción ejecutándose también es posible introducir en esta casilla un código de transacción que se ejecutará en el modo que corresponda. Por ejemplo,



Introducir esta secuencia implicará que se abandone la transacción que se está ejecutando en el modo y se ejecute la transacción RS01.

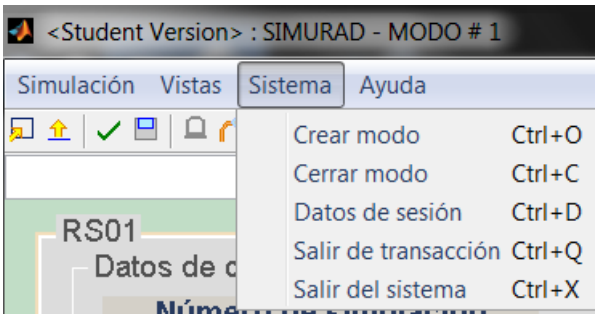
Ventana de transacción: Cuando se está ejecutando una transacción en un modo se muestra en esta ventana, indicando además sobre ella el título de la transacción.



Barra de notificación de mensajes: Por último para que el sistema notifique al usuario distintas situaciones o resultados de sus acciones en la barra que se encuentra debajo de la ventana de la transacción se muestran mensajes.

4.8.2.2. Gestión de los modos.

Las operaciones que se pueden realizar con los modos son crearlos, cerrarlos, cambiar entre uno u otro y ejecutar transacciones dentro de ellos. Para realizar estas operaciones, se pueden usar indistintamente, las opciones del menú (submenú sistema), los botones que aparecen a la izquierda de la barra de pulsadores, los comandos especiales /n /o en la casilla de comandos, etc. También es posible realizar estas acciones programáticamente.



Las operaciones que se realizan sobre un modo son completamente independientes de las operaciones que se realicen sobre otro. La sesión estará activa, mientras exista por lo menos un modo. Cuando se cierre el último modo de una sesión, se dará ésta por cerrada.

4.8.2.3. Implementación de los modos.

Para implementar la funcionalidad de los modos, se han desarrollados dos clases: el gestor de modos “blx_cl_momng”; y el propio modo en sí, “blx_cl_mode”. Además para gestionar la barra de notificaciones se utiliza la clase “blx_cl_msgmng”.

El gestor de modos se encarga de controlar la creación y destrucción de modos, llevar una lista de los modos existentes, controlar la asignación de transacciones a los modos y varias funciones más relacionadas con las anteriores. El gestor de los modos se debe utilizar en modo singleton, es decir una sola instancia por sesión.

Por contra, la clase asociada al modo, podrá tener múltiples instancias, uno por cada modo que se haya creado. Esta clase, guarda todos los datos relacionados con el modo, el número del modo, los controladores de los objetos gráficos del modo, los controladores de las funciones que permiten que el usuario interactúe con el modo, así como la referencia al objeto que gestiona la transacción que se ejecuta en el modo, y la referencia al gestor de mensajes del modo.

4.8.2.4. Gestor de mensajes del modo.

Como se ha explicado cada instancia de la clase que representa un modo, tiene asociado a su vez una instancia de la clase que representa al gestor de mensajes.

La clase “blx_cl_msgmng” es por tanto, la clase que gestiona la barra de notificaciones de cada modo. La función de la barra de notificaciones es permitir mostrar mensajes que informen al usuario de distintas situaciones.

Existen cuatro tipos de mensajes que se pueden mostrar al usuario, cada uno tiene su función:

“éxito” (Success): Este mensaje se debe mostrar cuando una operación en una transacción se ha realizado correctamente.

"advertencia" (Warning): Este mensaje se debe mostrar para advertir al usuario cuando algún dato introducido (o no introducido) aun no siendo erróneo, debería ser revisado con atención antes de proseguir con la operación de la transacción.

"error" (Error): Este mensaje se debe mostrar cuando el usuario ha introducido algún dato erróneo y debe por lo tanto cambiarlo para poder proseguir operando la transacción.

"terminación" (Abort): Este mensaje se debe mostrar cuando durante el procesado de las acciones del usuario sobre la transacción se ha llegado a un punto tal que es necesario interrumpirlo y cerrar el modo.

Para lanzar cada uno de estos mensajes en un modo específico se ha de llamar a uno de los siguientes métodos (un método por tipo de mensaje), una vez que se haya obtenido acceso a la instancia del modo en cuestión:

```
bl0_mt_smsg_show(bl0_ob_mode,bl0_st_mstxt)
bl0_mt_wmsg_show(bl0_ob_mode,bl0_st_mstxt)
bl0_mt_emsg_show(bl0_ob_mode,bl0_st_mstxt)
bl0_mt_amsg_show(bl0_ob_mode,bl0_st_mstxt)
```

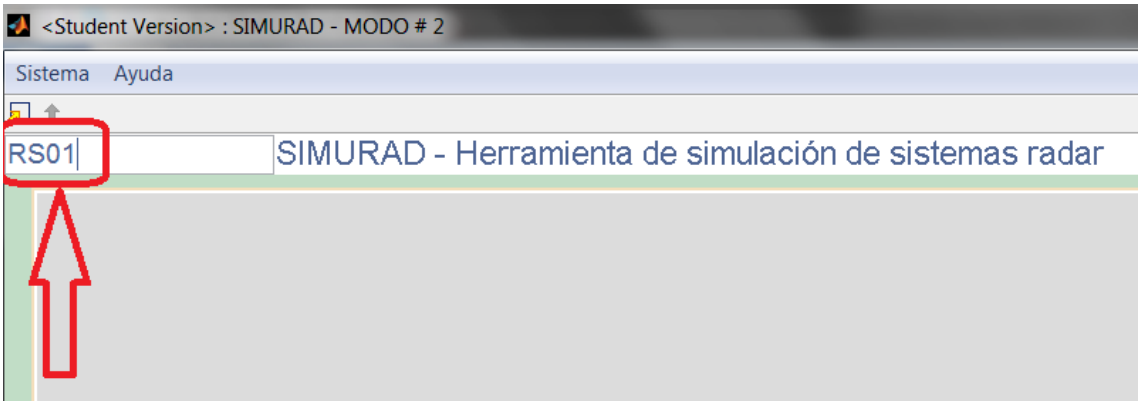
La variable "bl0_st_mstxt" deberá estar informada con el texto que se quiera mostrar en el mensaje.

4.8.3. LAS TRANSACCIONES.

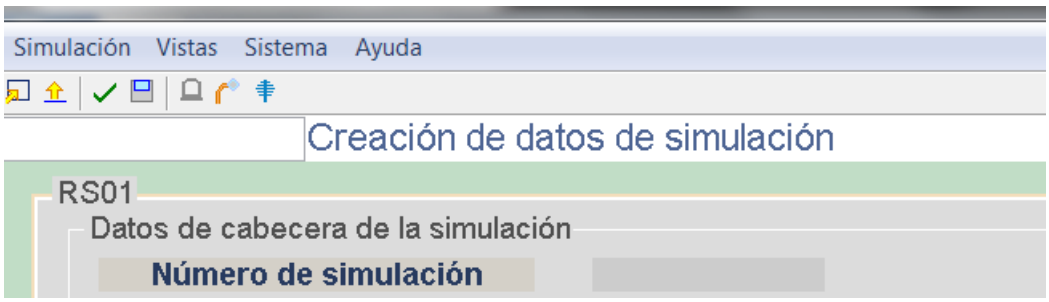
Las transacciones son las entidades sobre las que se basa y construye el entorno de simulación. Dentro de cada modo se puede ejecutar una sola transacción.

Cada transacción lleva asociado un código de transacción que actúa a su vez de identificador único. El registro principal de las transacciones disponibles en la simulación junto con sus principales características se gestiona mediante el mapa de datos "bl0_tcod".

El acceso a las transacciones se realiza tecleando su código en la casilla de comandos del modo.



Una vez que se carga una transacción en el modo, el título del modo cambia para reflejar la transacción seleccionada, así como se añaden los botones y las opciones específicas de menú de la transacción seleccionada a las que son propias del modo en sí.



También se puede ver que el código de la transacción se muestra en la esquina superior izquierda del recuadro donde se muestran los elementos gráficos de la transacción en sí.

4.8.3.1. Interacción de las transacciones con los modos.

El funcionamiento de las transacciones se ha diseñado de tal manera, que cada modo instanciado (referencia al objeto "bl0_cl_mode"), si tiene en un momento dado una transacción asociada, tendrá asociado una referencia a un "gestor de transacción".

Un gestor de transacción es una clase con ciertas características que permite realizar la gestión de la propia transacción en el modo, almacena todos los atributos que son propios de la transacción así como también los métodos que realizan las operaciones específicas de la transacción.

La relación entre un modo y su gestor de transacción se implementa mediante llamadas desde los métodos del modo a métodos del gestor de la transacción. Es decir todas las clases que se quieran utilizar como "gestor de transacción" tienen que tener unos métodos fijos con unas características determinadas que serán llamados por el modo cada vez que se necesite interactuar de alguna forma con la transacción.

Estos métodos que debe implementar cada gestor de transacción, se denominan "métodos de enganche" y son los siguientes:

- **bl0_mt_init_transaction**: En este método se deben inicializar todos los datos necesarios para la operación de la transacción.
- **bl0_mt_gui_layout**: En este método se deben crear los elementos gráficos propios de la transacción. Todos ellos se crearán dependientes de los propios elementos de orden superior del propio modo.
- **bl0_mt_gui_Callback**: En este método se gestionarán las interacciones del usuario con la transacción, como por ejemplo la introducción de datos.
- **bl0_mt_gui_ClickedCallback**: En este método se gestionarán otras interacciones del usuario con la transacción, por ejemplo, cliquear en un botón de la barra de pulsadores.
- **bl0_mt_gui_CloseRequestFcn**: Por último este método se utilizará para realizar aquellas operaciones que se deseen realizar cuando el usuario abandona la transacción, por ejemplo, salvado de datos, solicitud de confirmación al usuario, etc.

4.8.3.2. El gestor de datos.

Existe un paradigma de programación, conocido por sus siglas MVC, denominado modelo vista controlador (Model-View-Controller). Esta aproximación a la programación y diseño de interfaces con usuarios, se basa en separar la programación necesaria en dos capas bien diferenciadas que se encargan de funciones distintas. En toda herramienta, se pueden abstraer dos partes o componentes, la parte que se encarga de obtener y manipular los datos con que se trabaja y la parte que se encarga de presentárselos al usuario y gestionar sus interacciones.

Si se diseña bien una herramienta utilizando esta orientación se obtienen bastantes ventajas: Por un lado la sencillez de la implementación aumenta ya que al construir cada una de las partes solo tienes que preocuparte con una parte del problema, por otro lado, cualquier cambio en una de las partes, si los interfaces entre los dos se han definido de forma precisa, es totalmente transparente para la otra parte. Por ejemplo, una misma aplicación que permita a un usuario gestionar datos, puede pasar de utilizarse en una aplicación local a utilizarse en una aplicación web o incluso móvil, solo modificando la parte que implementa la interacción con el usuario. La parte que gestiona los datos no tiene porqué sufrir ninguna alteración.

Basándose en este modelo la herramienta de simulación se ha diseñado para permitir que cada "gestor de transacción" lleve asociado un "gestor de datos". La clase que implementa el gestor de transacción, se encargará de la interacción con el usuario mientras que el gestor de datos se encargará de los datos propiamente dichos.

Al igual que los modos interactúan con las transacciones mediante la implementación por los gestores de transacciones de ciertos métodos de enganche, los gestores de transacciones interactúan con los gestores de datos mediante la llamada en ciertos puntos específicos de unos métodos de "enganche" del gestor de datos.

Los métodos de enganche que debe implementar todo gestor de datos, son los siguientes:

- **bl0_mt_init_data**: Este método se utiliza para dar un valor inicial a los datos que maneja el gestor.
- **bl0_mt_load_data**: Este método se utiliza para informar los datos del gestor a partir de los datos almacenados de forma persistente (i.e. en la base de datos, pero en nuestro caso en el gestor de mapas).
- **bl0_mt_save_data**: Este método se utiliza para guardar los datos almacenados en el gestor de forma persistente(al igual que en el anterior, por ahora en los mapas del gestor de mapas).
- **bl0_mt_chk_data**: Este método se utiliza para verificar los datos informados en el gestor de datos, cada dato se contrastará contra sus características,
- **bl0_mt_get_data_values**: Este método permitirá recuperar el valor de uno o varios datos almacenados en el gestor a partir de su nombre.
- **bl0_mt_set_data_values**: Este método permitirá asignar valor a uno o varios datos del gestor, por cada dato que se desee informar, se especificará su nombre y su valor.

Como se ha mencionado cada gestor mantiene una serie de datos diferenciados por nombre y además una serie de características en modo de dominio, mapa de verificación, etc., a la que los datos deben atenerse. Tanto, los nombres de los datos como las características de

estas que maneja cada gestor se recuperan a partir de mapas de base de datos, permitiendo esto una gestión

Para homogeneizar todos los gestores de datos, se ha creado una clase modelo "bl0_cl_dmng" de la que todos los gestores de datos deben descender para heredar una funcionalidad común aparte de poder añadir características propias.

Para garantizar cierta uniformidad y facilidad en el desarrollo de los distintos gestores de datos, cada gestor de datos lleva asociado un nombre o código de gestor de datos, que en el mapa de datos bl0_dmcf tiene relacionados una serie de mapas con datos a partir de los cuales se determina la funcionalidad del gestor.

Cada gestor de datos, que implemente la clase "bl0_cl_dmng" tendrá a su vez asociados cuatro mapas:

- **Mapa de definición de datos**: En este mapa se especifican los datos que va a manejar el gestor junto con sus características.
- **Mapa de inicialización de datos**: En este mapa se especifican valores iniciales para los datos del gestor.
- **Mapa de carga de datos**: En este mapa se especifica la relación de mapas a partir de los cuales se cargaran datos en el gestor, así como el nombre de los métodos que realizarán dicha extracción.
- **Mapa de guardado de datos**: En este mapa se especifica la relación de mapas en los cuales se guardarán de forma persistente los datos del gestor así como los mapas que realizarán dicho almacenamiento.

Cuando un gestor de mapas se genera a partir de la clase "bl0_cl_dmng", el código de los métodos de enganche será el de esta clase, ya que es el que garantiza que se funcione de la forma configurada en los mapas especificadas arriba. No obstante, será posible si es necesario añadir alguna característica específica a un gestor, implementar el método deseado (esta operación se denomina redefinir) siempre que se llame a la implementación de la clase superior "bl0_cl_dmng" para garantizar el funcionamiento deseado.

4.8.3.3. El gestor de transacciones.

Si el gestor de datos se encarga de manejar los datos con los que el usuario interactúa, el gestor de transacciones se encarga de manejar la interacción del usuario con la interfaz gráfica, apoyándose en el gestor de datos correspondiente cuando sea necesario presentar datos al usuario o modificar los datos en función de las acciones del usuario.

Como los atributos y las características de un gestor de transacciones, dependen de una buena cantidad de variables y configuraciones, para unificar la creación de estos gestores se ha provisto un método central a modo de factoría, este método es el método estático "bl0_mt_get_tran_mng" de la clase "bl0_cl_tmng".

Una de los atributos comunes a todas las transacciones es el tipo de transacción. El tipo de transacción viene a reflejar cual es la operación que la transacción ayuda a realizar. Tipos comunes de transacción son: creación, modificación, visualización, etc. Utilizando el tipo de transacción se puede conseguir implementar varias transacciones con un mismo gestor de transacciones, cuyo comportamiento variará en función del valor del tipo de transacción que tenga informado. Por ejemplo, RS01, RS02, RS03 (Crear, modificar y visualizar datos de

simulación) son tres transacciones que utilizan el mismo gestor de transacciones pero que difieren en el valor que tienen en el tipo de transacción C, M, y V respectivamente (Tipo de transacción creación, modificación y visualización).

Al igual que para homogeneizar los gestores de datos se ha provisto una clase raíz de la cual pueden heredar un comportamiento común estos, para homogeneizar los gestores de transacciones también se prevé crear una serie de clases raíz que permitan mediante la herencia de sus características desarrollar fácilmente varias transacciones que posean características comunes. Para las transacciones de gestión de "objetos" de datos maestros, como por ejemplo, una antena, una trayectoria, una señal, etc., se ha creado la clase "bl0_cl_tmng".

4.8.3.4. El gestor de transacciones modelo para objetos de datos maestros.

A continuación, se explica cómo se ha diseñado el gestor de datos modelo para gestión de objetos de datos maestros, es decir, el implementado mediante la clase "bl0_cl_tmng". Los gestores de datos que se modelen según este patrón son apropiados para transacciones que gestionen objetos sencillos en las 3 modalidades ya mencionadas, creación, modificación y visualización.

El funcionamiento de este tipo de gestores se base en una secuencia de pasos que se reproducen entre la acción de un usuario y la presentación del interface gráfico al usuario tras esa función. También existe una primera fase que solo se ejecuta una vez y que se encarga de la creación y la configuración de los distintos elementos gráficos que integran la transacción.

El funcionamiento del gestor de transacción está gobernado por las configuraciones especificadas en una serie de mapas que van ligados a cada transacción. Estos mapas están asociados a la transacción en el mapa principal de las transacciones bl0_tcod.

Las tres fases principales en las que se divide el funcionamiento de este tipo de gestor de transacciones son:

'gc' -> Creación del interface gráfico (Gui Creation).

'bo' -> Antes de mostrar el interface (Before Output).

'ai' -> Después de mostrar el interface (After Input).

Cada una de estas fases del ciclo de funcionamiento tiene a su vez sus subfases y una complejidad asociada que se procede a explicar.

GC – Gui Creation.

Esta es la etapa en la que se implementa la creación y configuración de las características de los objetos gráficos que compondrán la transacción. Esta fase se ejecutará una sola vez, que será cada vez que se cargue la transacción en el modo.

A su vez está etapa, divide sus tareas en otras tres subfases que se detallan una por una.

CTC – Create Transaction Components.

Aquí se crearán los distintos objetos gráficos que compondrán la transacción, ahora bien, para garantizar un mismo aspecto y una alta reusabilidad, los objetos gráficos no se crearán directamente sino que deben crearse utilizando los conceptos de elemento gráfico y componente gráfico.

La mínima unidad de creación de objetos gráficos es el componente gráfico, que consiste en un objeto gráfico con una serie de características a precisar. Los distintos componentes gráficos llevan asociada una clave junto con unas características. Se gestionan mediante la clase bl0_cl_guic. Los componentes gráficos se configuran en el mapa bl0_guic. Un ejemplo de componente gráfico, es el "I1" que es una casilla para que el usuario introduzca datos.

Por encima de cada componente gráfico siempre habrá un elemento gráfico. Un elemento gráfico puede referenciar a un componente gráfico o a varios de ellos. Durante la creación de objetos gráficos siempre se ha de referir a los elementos gráficos. Al igual que los componentes gráficos, los elementos gráficos quedan identificados mediante una clave y una serie de características. La clase que gestiona los elementos gráficos es la bl0_cl_guie. Los elementos gráficos se configuran en el mapa bl0_guie. Un ejemplo de elemento gráfico es el "TM" que engloba todo lo necesario para crear el típico menú de transacción con sus opciones fijas.

Los elementos gráficos que componen la transacción se deben indicar en el método "bl0_mt_gge" que habrá que redefinir en cada clase de gestor de transacciones específicamente.

TTC – Transaction Type Configuration.

Después de haber creado los distintos objetos gráficos de la transacción, se pasa a configurar su visibilidad y disponibilidad en función del tipo de transacción. Esto responde a la necesidad de, utilizando un mismo gestor de transacciones para distintas transacciones, poder diferencia la disponibilidad de los distintos elementos. Por ejemplo, un campo que el elemento utilice para introducir una descripción debería ser editable solo para transacciones que permitan su edición, del tipo de creación o modificación, pero no para transacciones que permitan su consulta.

Todos los objetos gráficos que se creen para una transacción deben estar registrados en el mapa de objetos de esa transacción (a partir de aquí "mapa h"), el que se informa en el campo 'bl0_st_tmaph' del mapa 'bl0_tcod' para cada transacción.

En el mapa "h", cada objeto lleva especificado para cada tipo de transacción, si debe estar deshabilitado e invisible (N), si por el contrario debe estar habilitado y visible (V) o en un estado intermedio, visible pero no habilitado (E).

Es en esta etapa, donde el sistema efectúa estas configuraciones verificando además que cada objeto gráfico creado tenga su registro en el "mapa h".

IBO – Input Before Output.

Una vez que se han creado y configurado los objetos gráficos que conformarán la transacción será necesario un primer transporte de la información almacenada en el gestor de datos a los objetos gráficos que se la mostrarán al usuario para que pueda interactuar con ella. Estas tareas se realizan en esta etapa.

No se realiza directamente el transporte de los datos, sino que por así decirlo se indica para que objetos gráficos se va a requerir el transporte. En fases posteriores, se realizará el transporte a partir de lo indicado.

Existen dos mapas (asociados a la transacción en el mapa 'bl0_tcod' en los campos 'bl0_st_tmapb' y 'bl0_st_tmapv'), que llamaremos "mapa b" y "mapa v", que contienen la

configuración necesaria para poder determinar, en función de la información del gestor de datos, que datos mostrar en cada objeto gráfico, y que configuración deben tener estos objetos (que puede ser una lógica bastante más complicada que la simple discriminación por tipo de transacción).

En esta fase, todo objeto registrado en estos dos mapas se marca como "pendiente" de realizar este transporte.

BO – Before Output.

Esta es la etapa en la que se realizan todas las acciones necesarias para mostrar al usuario de la transacción en pantalla la información en base a la cual puede realizar las distintas acciones. Esta etapa se procesa cada vez que vaya a producirse un cambio en la información mostrada al usuario como consecuencia de una acción de este. Cada vez que el usuario ejecuta una acción se procesa primero la etapa AI (After Input) y luego inmediatamente después la etapa BO (Before Output). También se ejecuta una sola vez sin fase previa AI la primera vez que se carga una transacción, inmediatamente después de la fase GC (Gui Creation).

Esta etapa también está dividida en subfases en las que se realizan las distintas tareas asociadas con la presentación de datos al usuario.

MBO – Map Before Output.

En esta etapa se mueve la información existente en el gestor de datos que se debe mostrar al usuario del propio gestor de datos a los elementos gráficos que muestran esta información.

Esta etapa se basa en la configuración existente en el "mapa b" asociado al código de la transacción. En este mapa, por cada objeto gráfico que muestra información (representado por el nombre de su handler), se registran:

- El nombre del dato del gestor de datos a partir del cual se va a recuperar esa información.
- Para objetos complejos en los cuales la información se recupera a partir o en función de más de un dato, el nombre del método que realizará esta operación.
- El nombre de la rutina de conversión a la que podría estar sujeta la información contenida en el dato del gestor de datos antes de ser presentada en pantalla.[3]
- El nombre del método que se encarga de informar la información recuperada en el atributo que corresponda del objeto gráfico (el transporte propiamente dicho).

Como se ha indicado en la fase IBO, no se realiza el transporte para todos los objetos gráficos configurados en el "mapa b" sino para aquellos que son susceptibles de necesitar un transporte para refrescar la información que muestran. Por lo tanto, el sistema recorre los handlers de los objetos gráficos marcados como necesitados de transporte (en la primera ejecución de la transacción, todos), y ejecuta las siguientes acciones:

[3] Rutina de conversión: Existen ciertos datos que previamente a ser mostrados en pantalla o desde que son recuperados del valor introducido al valor con el que son almacenados, deben pasar por un proceso de conversión. Por ejemplo, algunos datos numéricos que se almacenarán siempre con un número fijo de ceros a la izquierda, no son mostrados en pantalla con los ceros por delante ni introducidos por el usuario así. Cada vez que el usuario los introduce por pantalla, el sistema añade los ceros necesarios antes de almacenarlos y viceversa, el sistema antes de mostrarlos por pantalla les elimina los ceros a la izquierda.
Para gestionar estos datos se han creado "rutinas de conversión" que no son más que una serie de claves que identifican conversiones de este tipo. Cada clave tiene asociado un método que permite procesar el dato previamente a ser mostrado y otro que permite procesar dato posteriormente a ser introducido por pantalla.
La gestión de las rutinas de conversión se realiza mediante la clase "blx_cl_conve".

1) Recupera la información del gestor de datos, ya sea a partir de lo indicado en uno de sus campos o con un método que la recupere de una combinación de ellos o un procesado complejo de lo indicado en uno solo).
2) Le aplica la rutina de conversión si así está configurado.
3) Informa el atributo del objeto gráfico que corresponda con el valor recuperado utilizando el método configurado, esta es la fase de transporte propiamente dicha).

CBO – Configure Before Output.

En esta otra etapa de la fase previa a la muestra del interfaz gráfico al usuario, se dan los atributos a los objetos gráficos que les correspondan en función de la información en el gestor de datos.

Al igual que en la fase "CBO", las tareas que se realizan en esta fase dependen de lo configurado para cada objeto gráfico en el "mapa v" en este caso. En este mapa, por cada objeto gráfico que sea susceptible de variar sus propiedades visuales en función de la información que se muestra, se registran (asociados al nombre de su handler):

- Un método que determina el valor de la configuración en función de la información contenida en el gestor de datos.
- Un método que traspasa el valor de la configuración al atributo que corresponda del objeto gráfico.

Análogamente a como ocurre en la fase MBO estas tareas no se llevan a cabo para cada objeto gráfico configurado en el mapa, sino que solo para aquellos que se consideran son susceptibles de sufrir algún cambio como consecuencias de cambios en los datos del gestor de los que dependen. También como antes, una primera vez sí que se ejecutan estas tareas para cada uno de los objetos gráficos configurados en el mapa.

Por lo tanto, el sistema recorre los handlers de los objetos gráfico para los que tiene previsto un cambio de configuración, y uno por uno:

- Determina con el primer método, el valor del atributo a transferir al objeto gráfico en función de los datos del gestor.
- Con el segundo método, informa el valor de recuperado en los atributos correspondientes.

RBO – Refresh Before Output.

Esta es una fase meramente administrativa, en la que se realizan las tareas de inicialización de todos los indicadores que gobiernan la presentación de los datos.

Como se ha explicado, las acciones de transporte de datos y configuración de los atributos, no se realizan para todos los objetos para los que existe una configuración asociado sino solo aquellos para los que existe una indicación de necesidad de este transporte. En esta fase se realiza el borrado de todos estos indicadores y se garantiza que todos ellos hayan sido tenidos en cuenta. Ésta, entre otras tareas del mismo tipo, son las que se realizan en esta fase.

NBO – Notify Before Output.

Esta fase se utiliza para mostrar al usuario las notificaciones en la barra de mensajes. Durante el procesamiento de las acciones del usuario o en cualquier fase del ciclo BO-AI, si se considera que se debe mostrar al usuario alguna notificación, los mensajes se van almacenando mediante llamadas al método "bl0_mt_addtmsg". En esta fase se muestran todos los mensajes almacenados de esta forma y se muestran en la barra de notificación de mensajes del modo.

OBO – Obligatory Before Output.

Entra la información que el usuario puede introducir por pantalla se puede desear que parte de ella sea de obligada cumplimentación. Esta configuración se almacena en el "mapa o" de la transacción. En este mapa, se registra una serie de objetos gráficos (siempre mediante el nombre de sus handlers) junto con el nombre del método que evaluará si el campo se puede considerar que ha sido debidamente cumplimentado.

El sistema resaltará de forma especial por un tiempo cada campo de obligada cumplimentación que no haya sido todavía debidamente informado por el usuario para llamar su atención sobre ello.

AI – After Input.

Esta, es la parte de procesamiento que se ejecuta inmediatamente después de que el usuario interactúe con la interfaz gráfica. Es decir, cada vez que el usuario interactúa con la interfaz gráfica, se ejecutará primera la etapa "AI" (After Input) y luego la etapa "BO". La etapa "AI" se encargará de gestionar las consecuencias de las acciones al usuario mientras que la etapa "BO" se encargará de modificar el interfaz gráfico que se muestra al usuario si este ha de variar como consecuencia de estas interacciones.

Al igual que las etapas "GC" y "BO" la fase "AI" se divide a su vez en subetapas (dos):

MAI – Map After Input.

Esta fase se encarga de transportar los datos introducidos por el usuario en el interfaz gráfico a la capa del gestor de datos. Se puede decir, que hace un trabajo contrario al que hace la fase MBO, que transporta los datos del gestor de datos a la interfaz gráfica. El transporte de los datos al gestor de datos implica que éstos serán contrastados contra las características y atributos que se les han dado en el gestor, si el contraste resulta negativa, el usuario es informado de que dato introducido no es correcto y debe revisar.

Al igual que otras fases, el funcionamiento de esta fase depende de lo configurado en uno de los mapas de la transacción, en este caso, el "mapa a". Mientras que los mapas de las fases "BO" tenían todos sus registros referenciados a los nombres de los "handlers" de los objetos gráficos; en el caso de los mapas que gobiernan el funcionamiento de las fases "AI", las claves de registros son las etiquetas de los objetos gráficos. Una de las propiedades de los objetos gráficos, es que se les puede asociar una ID o "tag" que denominaremos etiqueta. Cada objeto gráfico susceptible de interacción tendrá asociada su etiqueta.

Es a cada una de estas etiquetas que se le asocian los registros del "mapa a", llevando cada registro la configuración necesaria para procesar la información introducida por el usuario en el objeto gráfico con dicha etiqueta. Por cada etiqueta, se registra por lo tanto la siguiente información:

- El nombre del dato del gestor de datos sobre el que se "volcará" la información introducida por el usuario.
- En el caso de que la información introducida por el usuario afecte a más de un dato del gestor de datos o de que la extracción de esta información no se pudiese obtener directamente del valor de un atributo del objeto gráfico, también se puede indicar el nombre de un método que realice esta labor de extracción.

- Al igual que ocurre en la fase MBO la información introducida por el usuario puede estar sujeta siempre a una serie de transformaciones. En ese caso, se puede indicar aquí la clave de la rutina de conversión que realizaría estas transformaciones.
- Por último se indica el método que recoge del objeto gráfico la información introducida por el usuario.

Un proceso importante que ocurre en esta fase es el "marcado" de los distintos datos del gestor que han sido afectados por la interacción del usuario. Cada dato del gestor al ser modificado marcará los correspondientes "handlers" de los objetos gráficos que a su vez obtienen sus características o información a partir de ese dato. De esta forma cuando se ejecute la fase "BO" solo se procesarán aquellos "handlers" de objetos gráficos cuyas características hayan podido verse afectadas por modificaciones en los datos.

CAI – Callback After Input.

Esta la otra fase que se encarga de la gestión de las acciones del usuario, mientras que la fase "MAI" se encarga de gestionar los datos introducidos por el usuario, esta fase se encarga de gestionar las acciones del usuario que no consistan en introducir datos. Ejemplos de estas acciones, son pulsar botones, seleccionar opciones de menú, etc.

Al igual que en la otra fase, el procesado que se realiza en esta fase depende de lo configurado en un mapa asociado a la transacción, en este caso, el "mapa c". También, los registros de este mapa tiene como clave el nombre de etiquetas que se asocian a los objetos gráficos cuya utilización por parte del usuario se prevea que puedan desencadenar una "acción" o procesado posterior.

Cuando el usuario acciona un elemento gráfico, como por ejemplo, un botón, el sistema al llegar esta etapa recupera la etiqueta asociada a este elemento, y luego busca si hay algún registro en el "mapa c" asociada a este etiqueta. Si existe este registro, este llevará informado el nombre de un método que es el que se ejecutará para realizar las acciones que se hayan previsto.

4.8.3.5. Componentes de las transacciones.

Una de las características que se ha buscado en la implementación del entorno, es la reusabilidad de los objetos desarrollados, no solo por la economía de tiempo que ofrece, sino para que el usuario se acostumbre a tratar siempre con elementos de la misma apariencia y no tenga que estar constantemente aprendiendo a utilizarlos. En esta filosofía de han desarrollado varios objetos algunos de los cuales se explican a continuación.

- El visualizador de listas.

El visualizador de listas, o MLV (Matlab List Viewer) es una herramienta que se utiliza para mostrar al usuario datos en forma de tabla. Además de los datos, también incorpora una serie de herramientas para tratar estos datos, como por ejemplo la posibilidad de ordenarlos.

El MLV se ha implementado mediante la clase "bl0_cl_mlv" y se debe utilizar llamando a sus APIS para pasarle la información necesaria de tanto de configuración como de datos para mostrar. Si cada vez que se quiere mostrar información de tipo tabla se utiliza este elemento, se conseguirá con un mínimo esfuerzo de desarrolla disfrutar de toda la funcionalidad que ofrece así como de la futura funcionalidad que podría incorporársele.

- El buscador de datos.

El buscador de datos, es otra herramienta, que en este caso se utiliza para que el usuario pueda buscar datos específicos. Está diseñado para ofrecer al usuario la posibilidad de restringir la búsqueda rellenando una serie de criterios de selección, mostrarle los resultados de la búsqueda con información relacionado en una tabla utilizando el MLV, y permitirle seleccionar de entre los registros mostrados, uno para continuar el proceso. El buscador de datos (sus funciones esenciales) está implementado mediante la clase "bl0_cl_sutil".

Tanto los criterios de selección como los resultados buscados, como la propia búsqueda en sí, son totalmente configurables y se implementan creándose una clase llamada "alimentadora" que herede de la clase "bl0_cl_sfeed".

Una de las aplicaciones derivadas es el selector de valores de objetos, que proporciona la posibilidad de seleccionar un valor de un mapa de datos del gestor de mapas mediante un pop-up, en este pop-up se puede buscar por clave o por la descripción asociada a la clave. Para disponer de toda esta funcionalidad basta con indicar el nombre del mapa. Esta aplicación se implementa con la clase "bl0_cl_ovsel" que es una clase derivada de "bl0_cl_sutil".

4.9. EJEMPLOS.

A modo de ejemplos de desarrollo en este entorno de simulación se muestran algunas transacciones que ya han sido desarrolladas, estas transacciones no se consideran ni mucho menos completas sino que son susceptibles de incorporar muchas más funcionalidades siguiendo las directrices indicadas.

4.9.1. MENU INICIAL DE LA SIMULACIÓN.

El entorno de simulación consistirá (esperanzadoramente) en una cantidad grande de transacciones que el usuario ejecutará para mantener todos los datos maestros y transaccionales que serán necesarios, así como para visualizar los distintos resultados.

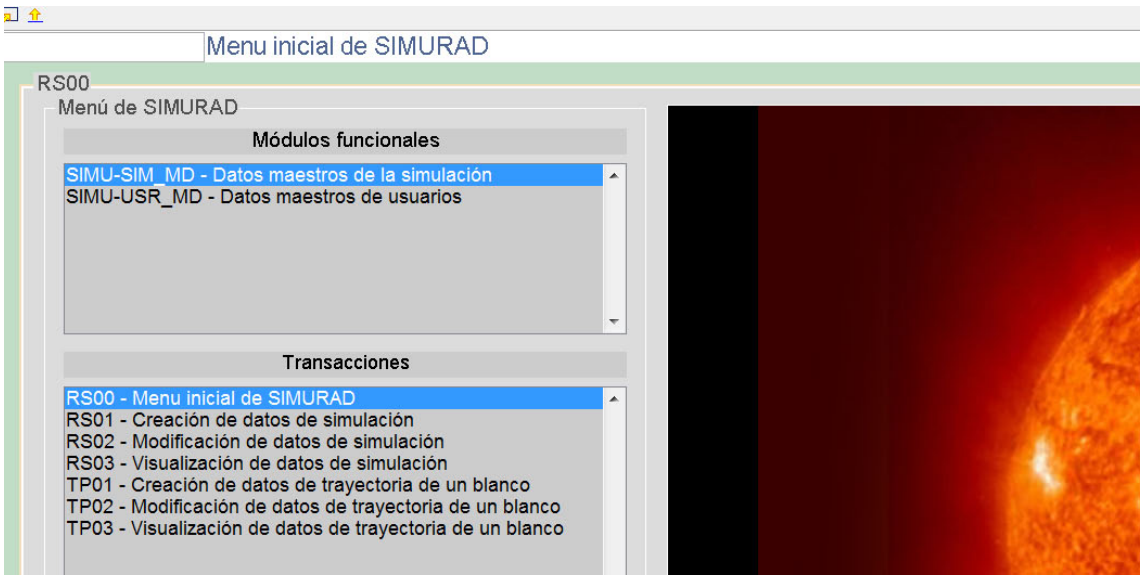
Para que el usuario pueda acceder a las distintas transacciones de una forma sencilla y ser consciente de las transacciones disponibles, se ha creado la transacción RS00, que permite acceder a las distintas transacciones habilitadas a partir de un menú desplegable en el que estas se muestran. Pulsar sobre una de ellas, provocará que la transacción se habrá en un modo adicional.

Debido a lo útil que representa partir de esta transacción como transacción base, se ha configurado como transacción por defecto para el usuario ADMIN, y bien podría serlo para todos los demás usuarios.

Para poder clasificar las transacciones y ayudar al usuario a localizar las que busca se ha creado el concepto de módulos funcionales. Los módulos funcionales agrupan transacciones que realizan funciones relacionadas. En la transacción RS00 las transacciones existentes se muestran en un árbol que tiene como nodos los módulos funcionales.

Los módulos funcionales se configuran en el mapa "bl0_modf" y cada transacción se asigna a su correspondiente módulo funcional en el mapa "bl0_tcod", que es el mapa principal de configuración de las transacciones.

A continuación se muestra el aspecto que presenta la transacción:



4.9.2. GESTIÓN DE LAS SIMULACIONES.

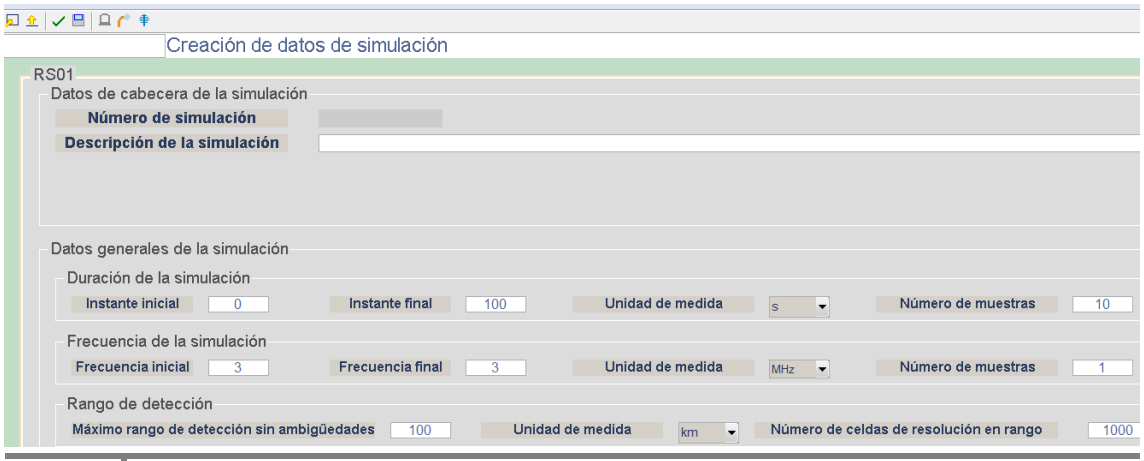
Una de las transacciones centrales al entorno de simulación de radar, debe ser la que gestione los datos y las configuraciones de la simulación en si (se considera útil separar la presentación de los resultados en una transacción aparte). Para la gestión de los datos de la simulación se han esbozado las transacciones RS01, RS02, RS03.

El que se hayan creado tres transacciones, responde a las tres formas en que a priori se puede interactuar con los parámetros de una simulación: crear (1), modificar (2) y visualizar (3).

Como bajo el concepto de simulación se deben recoger parámetros relacionados con distintos módulos funcionales, se ha creado la transacción de modo que por un lado se tenga acceso a los datos centrales de la simulación (que no corresponden a un módulo específico) y por otro lado a los datos correspondientes a cada módulo de forma diferenciada.

Para guardar los datos de las simulaciones se utilizarán una serie de mapas de datos todos relacionados entre sí. El mapa principal será el mapa “bl0_simh” que será el que guarde en cada registro los datos de una simulación distinta. La clave de este registro será por supuesto, el número de simulación. Luego, existirá además por cada módulo un mapa adicional, por ejemplo “bl0_sim1” será el mapa donde se guarden los datos específicos del módulo donde quedan englobados los parámetros correspondientes a la trayectoria del blanco.

A continuación se muestra el aspecto que presenta la transacción:



4.9.3. TRAYECTORIAS DEL BLANCO.

Como ejemplo de transacción de datos maestros, se han creado una serie de transacciones para gestionar los parámetros relacionados con la trayectoria de un blanco.

Al igual que anteriormente, para gestionar estos datos, se habilitan, no una sino tres transacciones: TP01 (crear trayectoria), TP02 (modificar trayectoria), y TP03 (visualizar trayectoria).

Esta transacción se ha implementado con un gestor de transacciones basado en el patrón “bl0_cl_tmng”, que es el que se ha explicado que debería servir de base para todas las transacciones que gestionen los datos maestros de un objeto específico (en este caso trayectoria).

Para guardar los datos de la trayectoria, inicialmente se utiliza solo el mapa “bl1_path”, aunque según se fuesen añadiendo más parámetros podría ser necesario y/o conveniente apoyarse en mapas auxiliares.

A continuación se muestra el aspecto que presenta la transacción:

CAPíTULO 5. GUÍA PARA DESARROLLOS SUBSIGUIENTES.

En este capítulo, se esbozarán por último a modo de guía, una serie de funcionalidades que se pueden incorporar a la simulación según se vaya disponiendo de recursos y tiempo, como se ha explicado, el desarrollo realizado, por motivos de tiempo, no llega a ser más que el "esqueleto que servirá para sostener la carne"; por lo que también es necesario añadir una cantidad importante de trabajo adicional para obtener un software para un entorno de simulación de radar.

Funcionalidad por completar.

Obviamente, el núcleo de trabajo a realizar es añadir las transacciones que permitan gestionar los datos maestros y transaccionales, que permitan caracterizar de forma apropiada (y útil) una simulación de detección y seguimiento de un blanco de radar. Este trabajo es a priori infinito si se enfrenta con ambición por que siempre habrá un parámetro adicional que añadir. Por lo tanto, el mejor enfoque, es una pieza a la vez, es decir, añadir una característica primero, luego otra y así sucesivamente. Hay que evitar mientras se hace esto perder una visión global que permita que al final cada pieza se ajuste perfectamente al todo. Una forma de hacer esto, puede ser el centrarse en módulos funcionales. Una persona, podría por ejemplo, encargarse de añadir toda la funcionalidad relativa a las características de la antena.

Una parte, que bien pudiere ser la más importante, y que queda por desarrollar es la relativa al cálculo y presentación de los resultados. Una vez que el usuario ha registrado correctamente todos los parámetros, es necesario, a partir de ellos calcular los resultados de la simulación. Esto es, una tarea, que sin duda tiene asociada una complejidad importante. Pero, aún más esfuerzo conllevará la tarea de mostrar los resultados al usuario. Para no desperdiciar el potencial y el esfuerzo anterior, la presentación de resultados debería ofrecer al usuario el máximo de funcionalidades y posibilidades de elegir entre distintas modalidades de presentación. (El propio Matlab presenta varias herramientas para la muestra de gráficos que deberían ser explotadas).

Funcionalidad base adicional.

Existen una serie de aspectos que se deben desarrollar para mejorar las prestaciones del entorno de simulación.

Uno de las cosas que queda por implementar es un complemento al diccionario sintáctico de datos, un diccionario semántico. Un diccionario semántico, asociaría al nombre de un dato, unas descripciones que especificasen cuál es el significado del dato y además unas características físicas que se especificarían mediante la asignación de un dominio. De este modo, se podrían tener, por ejemplo, el dato "número de trayectoria" y el "dato número de simulación" ambos recaer sobre un dominio "contador de diez"; i.e. dos datos que compartiesen la misma especificación física podrían ser definidos de forma distinta por su significado. Esto disminuiría con mucho el número de dominios a mantener, y aportaría mucha claridad y estructuración al desarrollo. Además siempre que se necesitase mostrar gráficamente un dato al usuario su descripción podría tomarse del diccionario semántico. Por supuesto, una vez que el diccionario semántico estuviese disponible, todas las verificaciones de variables se harían contra este diccionario en vez de contra el sintáctico (el cual seguiría activo de forma indirecta porque cada dato del diccionario semántico tendría asignado un dominio del diccionario sintáctico).

Otro concepto importante a añadir, es el concepto de bloqueos de utilización. El entorno de simulación idealmente debería permitir su utilización recursiva y simultánea por varios usuarios. Cuando esto ocurra, es necesario evitar que varios usuarios gestionen los mismos datos a la vez porque esto puede llevar a confusiones e inconsistencias varias. Para controlar este problema será necesaria identificar cada una de los posibles objetos de la simulación (antenas, rutas, señales) y llevar un registro de cada objeto que se esté tratando. El sistema no debería permitir tratar un objeto específico, por ejemplo, la ruta número 10, si tiene constancia de que ya está siendo gestionada por otro usuario.

Otra faceta a incorporar a este tipo de entorno es el concepto de autorizaciones, según el sistema crezca en complejidad será necesario diferenciar las tareas que realizan en él los distintos tipos de usuarios. Esto se puede conseguir asociando a las distintas interacciones con el sistema distintas verificaciones, y luego agrupando conjunto de autorizaciones para estas verificaciones en "roles" que finalmente y por último se asignarán a los usuarios para determinar que pueden y no pueden hacer. Añadir verificaciones de autorización a las distintas acciones que se ejecutan sobre el entorno de simulación conllevará un esfuerzo importante de desarrollo, pero sin duda, serán necesarios según la complejidad del sistema crezca y sea necesario diferenciar perfiles de usuarios.

Una ayuda importante al usuario del entorno que mejorará sin duda su experiencia de uso, será la incorporación de ayudas de búsqueda o "matchcodes" a aquellos campos que se le presenten en la interface gráfica y que solo pueden tomar unos valores específicos. Pulsando un icono de aspecto reconocible, al lado de cada campo con ayuda de búsqueda asociada al usuario debería mostrársele un pop-up para que pueda elegir el valor que desea para el campo. Si la selección es complicada por el número de posibilidades, también debería incluirse la posibilidad de influenciar la búsqueda mediante la introducción de criterios de selección. Todo esto debería ser implementado y encapsulado de forma que el añadir una ayuda de búsqueda a un campo requiriese un trabajo mínimo, el mínimo para especificar los datos entre los que se elige. Una buena idea para simplificar la asociación de ayudas de búsqueda, es relacionar cada dato del diccionario semántico con su propia ayuda de búsqueda.

También aportará bastante a la gestión y mantenimiento del entorno de simulación la inclusión de transacciones específicas para el mantenimiento de los distintos componentes llamemos "de base". Por ejemplo:

- Transacciones para consultar (y modificar), las características de los mapas de datos. –
- Transacciones para consultar la información contenida en los mapas de datos (y que también permitan por supuesto insertar, modificar o borrar registros).
- Transacciones para consultar/actualizar el diccionario semántico.
- Transacciones para consultar/actualizar el diccionario sintáctico.
- Transacciones para consultar/actualizar el maestro de usuarios.
- Transacciones para consultar/actualizar las características de las propias transacciones.
- Transacciones para consultar/actualizar los roles con sus distintas autorizaciones.

En definitiva, muchas de las configuraciones que inicialmente solo se pueden actualizar modificando líneas de código, deberían poderse actualizar mediante una combinación de información guardada en mapas y unas transacciones que permitan actualizar de forma gráfica esos mapas.

Otra característica que será necesario incorporar al entorno de simulación según la complejidad de éste vaya incrementándose, es algún tipo de organización de los objetos de desarrollo, así como la creación de varios entornos, de forma que por ejemplo, haya un entorno de desarrollo, otro de pruebas y otro productivo donde se trabaje. Los nuevos desarrollos se implementarían en el entorno de desarrollo y se irían transportando sucesivamente a pruebas y luego a productivo. Esto permitiría una disponibilidad permanente del entorno de simulación y un control sobre las mejoras y correcciones que se fuesen añadiendo.

Funcionalidad por sustituir.

Como se ha apuntado en otras partes, lo que se ha construido no es más que un prototipo o esbozo de lo que tiene que ser el entorno de simulación. Para conseguir un entorno de simulación que realmente pueda ser utilizado de forma productiva es conveniente implementar los siguientes cambios en la tecnología:

- Sustitución del almacenaje persistente de datos en ficheros por almacenamiento en base de datos relacional (o sistema equivalente). Esto no solo aumentará con mucho la velocidad de procesado de esta capa sino que facilitará que varios usuarios accedan de forma concurrente a los mismos datos desde distintos sitios.
- Sustitución de la arquitectura. Ejecutar en el propio programa de Matlab localmente en un pc, el entorno de simulación, no es lo más óptimo. No solo sufre las limitaciones de procesamiento que un pc pueda tener (que son visibles incluso en este estado embriónico del entorno), sino que no permite la utilización del entorno de simulación desde varios puntos de acceso. Se debería pasar a un entorno cliente servidor, de tal forma que existiese un cliente sencillo que los usuarios del entorno deberían instalar y que conectaría con un servidor donde se ejecutaría el núcleo de la funcionalidad del entorno, el servidor sería el que accediese la base de datos.
- Sustitución del propio lenguaje de programación, el Matlab, por uno más adecuado y eficiente en las funciones de interfaz con el usuario y procesado y almacenado de los datos maestros y transaccionales. El código en Matlab solo debería quedarse para implementar la funcionalidad relativa al procesado de las señales y datos de las simulaciones de radar en si (que es para lo que está diseñado).

CAPíTULO 6. BIBLIOGRAFÍA.

Introducción teórica a los diferentes aspectos de un sistema radar.

Introduction to Airborne Radar. Second Edition. George W. Stimson. Second edition(1 Diciembre, 1998). SciTech Publishing.

Simulaciones de sistemas radar en Matlab.

Radar Systems Analysis and Design using Matlab. Third Edition. Bassem R. Mahafza (20 Mayo, 2013). Chapman and Hall (CRC).

Radar Signal Analysis and Processing Using Matlab. Bassem R. Mahafza(19 Junio,2008). Chapman and Hall (CRC).

[i] SAP (Acrónimo para Sistemas Aplicaciones y Productos) es una empresa alemana fundada en los años 70 por ex ingenieros de IBM, que ha llegado a convertirse en una multinacional cuyo producto software para empresas la convierte en la segunda empresa de software más grande del mundo después de Microsoft. Su producto principal y más exitoso es su ERP (Enterprise Resource Planning) que es un software que permite gestionar de forma integrada todas las actividades de una empresa.

[ii] Para facilitar la lectura del código, se ha decidido utilizar una serie de convenciones para la nomenclatura de los distintos objetos de desarrollo. A continuación se detallan las distintas normas:

Clases: Se utilizará un prefijo inicial seguido de un guion bajo, luego el par de caracteres "cl" seguidos de otro guion bajo y, por último, una serie de caracteres que identificarán la clase. El prefijo inicial fija el módulo de funcionalidad con el que la clase está relacionada y el "cl" indica que se trata de una clase. Por ejemplo, blx_cl_dadom, es el nombre de una clase relacionado con el bloque X (funcionalidad común a todos los módulos), y "dadom" es una abreviatura de data domain.

Dominios: Se utilizará un prefijo inicial seguido de un guion bajo, luego un par de caracteres seguidos de otro guion bajo, y por último, una serie de caracteres que identificarán el dominio. El prefijo inicial indica el módulo de funcionalidad con el que el dominio está asociado. El par de caracteres identificarán el tipo de dato (ver variables). Por ejemplo, en "bl1_st_patnr", "bl1" indica que este dominio está asociado al bloque 1 de blancos, "st" indica que se trata de un string y "patnr" es una abreviatura para el path number o número de trayectoria.

Mapas: Se utilizará un prefijo inicial seguido de un guion bajo y luego una serie de caracteres que identificarán el mapa. El prefijo inicial indicará el módulo de funcionalidad con el que el mapa está relacionado. Por ejemplo, en bl0_tcod, bl0 indica que pertenece al módulo general y tcod es una abreviatura para transaction codes, este mapa guarda la configuración de las distintas transacciones.

Variables: Las variables que se utilizan en las secciones de código están sujetas a las siguientes convecciones, un carácter que indica su función dentro del segmento de

código seguido de un guion bajo, un par de caracteres que identifican el tipo de dato seguidos de otro guion bajo y por último, una serie de caracteres que sirvan para identificar lo que representa la variable.

El primer carácter puede tomar cualquiera de los siguientes valores:

- "i": Parámetro de entrada a una función o método.
- "e": Parámetro de salida a una función o método.
- "c": Parámetro de entrada/salida a una función o método (de cambio).
- "l": Variable local dentro de una función o método.
- "g": Variable global.

Los dos caracteres que indican el tipo de dato pueden tomar cualquiera de los siguientes valores:

- "xx": Cualquier tipo de dato.
- "st": String o secuencia de caracteres.
- "ob": Referencia a objeto
- "nu": Número.
- "mp": Mapa de datos(referencia a containers.Map)
- "ce": Celda(s).
- "sc": Estructura.
- "sa": Array de estructuras.
- "na": Array numérico.
- "ca": Array de caracteres.
- "sy": Símbolos (fórmulas).

Por ejemplo, "l_sc_maval", corresponde al nombre de una variable local, que sea una estructura y "maval" es una abreviatura de map values.

Variables (Parámetros): La nomenclatura para las variables utilizadas como parámetros en los métodos, cuando se define un método, es la misma que normalmente, salvo que el primer carácter se sustituye por un prefijo identificativo del módulo al que va asociado el parámetro. Por ejemplo, si la variable "l_sc_maval" se utilizase en la declaración de un método se utilizaría "bl0_sc_maval" para indicar que es una variable característica del módulo general, el "bl0" habría sustituido a la "l" de local.

Atributos (de clases tipificadas): La nomenclatura de estos atributos es la misma que para las variables utilizadas como parámetros.